UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

HEALTH ENGINEERING

MAJOR IN BIOMEDICAL ENGINEERING

CARDIOVASCULAR DISEASE PREDICTION THROUGH ARTIFICIAL INTELLIGENCE ALGORITHMS

PREDICCIÓN DE ENFERMEDADES CARDIOVASCULARES MEDIANTE ALGORITMOS DE INTELIGENCIA ARTIFICIAL

Submitted by:

Elisa Juliá Martínez

Tutored by:

Ezequiel López Rubio

Rosa María Maza Quiroga

Department:

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

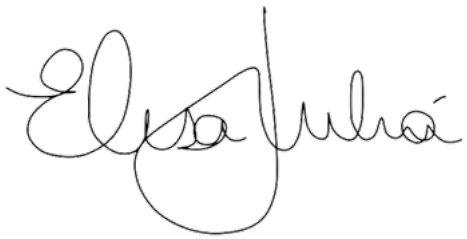
June 2020

Declaration of originality

I, Elisa Julia Martinez, ID number 32093964V, student pursuing the Bachelor Degree in Health Engineering, major in Biomedical Engineering by the University of Malaga,

hereby declare:

The work titled *"Cardiovascular Disease Prediction Through Artificial Intelligence Algorithms"* is my own work and that I have not sought or used inadmissible help of third parties to produce this work. I have fully referenced all sources and used inverted commas for all text directly quoted. This work has not yet been submitted to another examination institution – neither in Spain nor outside Spain – neither in the same nor in a similar way and has not yet been published.



Elisa Julia Martinez

Malaga, June 2020

Abstract

Artificial Intelligence (AI) has been around for a few decades now, and the potential applications in medicine have not been overseen. There are many people working worldwide to implement safe technology into the medical workflow to aid clinicians and diagnosticians make more informed decisions. This project focuses on the implementation of 5 different classifier algorithms, analyse how these adapt to the provided data and are able to predict values, and then create a genetic algorithm that detects which combination of parameters for each classifier gives out the best results, in terms of accuracy. The five classifiers were implemented using Python's *sci-kit learn* package and for the genetic algorithm no additional package was used. The results showed that some algorithms adapt better when undergoing evolution, meaning the accuracy increases as generations go through. Other algorithms showed a decrease in their performance, which suggested that for each type of classifier, a further study is needed into the effect of each parameter and the importance of values that for this project were considered constants, such as number of generations, number of individuals on each generation, probability of mutation and cross-over, sample size and size of training, validation and testing sets.

Keywords: Artificial Intelligence, Cardiovascular Disease Prediction with Machine Learning, Genetic Algorithm, Classification, Python, Sci-kit learn.

Resumen

La Inteligencia Artificial (AI) ha sido un tema muy hablado en las últimas décadas, y sus potenciales aplicaciones en la medicina han siempre sido reconocidas. Hay mucha gente trabajando para implementar la tecnología disponible en el flujo médico de manera segura, para ayudar a médicos y personal sanitario a tomar decisiones más informadas. Este proyecto se centra en implementar 5 algoritmos distintos de clasificación, analizar cómo se ajustan a los datos disponibles y, posteriormente, crear un algoritmo genético que detecte la combinación de parámetros de cada algoritmo, con la que se obtienen mejores resultados, medidos a través de la exactitud. Los algoritmos fueron implementados con la librería *sci-kit learn* de Python; para el algoritmo genético no se utilizó ninguna librería adicional. Los resultados mostraron que algunos algoritmos se adaptan mejor a la evolución, es decir, la exactitud aumenta con el paso de las generaciones. Otros algoritmos mostraron una disminución de este valor, sugiriendo que se necesita estudiar para cada tipo de algoritmo el impacto de cada parámetro, además de los valores que en este proyecto se consideraron constantes: número de generaciones, número de individuos por generación, probabilidad de mutación y cruce y el tamaño del conjunto de datos y de los subconjuntos de entrenamiento, validación y pruebas.

Palabras clave: Inteligencia Artificial, Predicción de enfermedades con Machine Learning, Algoritmo Genético, Clasificadores, Python, Sci-kit learn.

Acronyms

AI Artificial Intelligence

AUC Area Under Curve

CHD Coronary Heart Disease

CM Confusion Matrix

CS Circulatory System

CVD Cardiovascular Disease

DT Decision Tree

EHR Electronic Health Record

FN False Negative

FP False Positive

FPR False Positive Rate

GA Genetic Algorithms

HIS Hospital Information Systems

KNN K-Nearest Neighbour

ML Machine Learning

MLP Multi-Layer Perceptron

PACS Picture Archiving and Communication Systems

RF Random Forest

ROC Receiver Operating Characteristics

SVM Support Vector Machine

TN True Negative

TP True Positive

TPR True Positive Rate

WHO World Health Organisation

Contents

List of Figures

List of Figures

List of Tables

1. Introduction

1.1. Motivation

The process of treating a patient, from the moment an anomaly is detected, through diagnosis, treatment and follow up, is extremely sensitive and needs to be monitored thoroughly, making it very resource consuming: unfortunately, not all health centres are able to fully undertake these tasks due to lack of resources. Thousands of people are working restlessly to improve healthcare systems and reduce the strain they are constantly under. One resource that has shown great potential amongst the many being developed, is computer aided decision making: "Rule-based approaches saw many successes in the 1970s, and have been shown to interpret ECGs, diagnose diseases, choose appropriate treatments, provide interpretations of clinical reasoning and assist physicians in generating diagnostic hypotheses in complex patient cases" (Yu, Beam, & Kohane, 2018). Automatising (to some extent) these processes has the potential of having a great positive impact on healthcare systems: "AI applications also have the potential to bring clinical expertise to remote regions where specialists are scarce or not available" (Yu et al., 2018).

There are some health conditions that can benefit greatly form applying semi-automatised support systems. Some of these are cardiovascular diseases (CVDs) for being amongst the most common conditions nowadays and a major cause of death worldwide. According to the World Health Organisation (WHO), 17.9 million people die each year of heart related conditions, which makes 31% of all death's worldwide (WHO, 2019).

There are different types of CVDs and different risk factors that contribute to the

presence and extent of the disease. Being such common conditions, the relation between the risk factors and the disease are well known, which allows (semi)automatic systems to aid in the process very successfully.

Hence, the topic and scope of this project arose from the extent in which AI is impacting healthcare worldwide, and the immense effort being put into developing ever-evolving programmes that adapt to our systems needs. This work focuses on CVDs mainly because of their frequency and the impact throughout the globe.

1.2. Objectives

The main objective of this project is to develop a programme capable of predicting whether a patient will suffer from cardiovascular disease (in general terms), using different algorithms within the AI range. The aim is to classify the individuals with the highest possible accuracy, using Supervised Machine Learning.

To do so, 5 different intelligent algorithms have been configured, tested and compared, to see which adapts better to the provided data. To further explore the configuration of each algorithm and the effect it has on the accuracy, a genetic algorithm has been developed to analyse variability and performance, thus finding the configuration for each of the 5 algorithms that returns the best results.

1.3. State of the Art

Disease prediction is not the only use of AI in the medical field, it in fact is just one in the extensive range of applications people are working on worldwide. Great potential for AI is being discovered continuously, and it is being gradually included in the daily medical workflow. "AI methodologies are now commonly used to aid in computer vision, speech recognition, and natural language processing (NLP). (...) AI has also been used in analyzing clinical data, including medical images, electronic health records (EHRs), and

physiological signals" (Wang & Preininger, 2019).

Perhaps the biggest restriction AI has when applied to medical workflow is the availability of information. Although there is data being digitised continuously, there are still many countries that preserve most of their medical records on paper. This issue, despite being of great importance, has not stopped researchers from developing methods to use the information readily available in digital formats. Not only data must be loaded into Hospital Information Systems (HIS), but the already digitised information must be integrated with data coming from many different sources and in many different formats, such as Electronic Health Records (EHR), clinician's notes (in "plain text"), images in DICOM, etc. Additionally, the system must be able to recognise different time frames, since clinical decisions and diagnosis must almost always contemplate a full patient's overview, rather than just one episode. Bettencourt-Silva, Mannu, and Iglesia published a methodology to represent all the information from the EHR of a patient into a single clinical-pathway (2016).

Important to note that the aim of all AI systems is to aid clinical decision making, becoming support for diagnosticians to speed up the process or act as a second opinion. It is never creating diagnoses by itself, without human supervision. "The machine learning systems are not to replace doctors or make absolute decisions in a patient's treatment. (...) [Doctors] want to use the artificial intelligence to support decisions and make recommendations based on the findings" (Martin, 2019). In fact, Holzinger mentioned *interactive Machine Learning (iML)* as a way to implement ML algorithms when not sufficient training data is available, thus needing aid of an agent (human or not) and making use of human cognitive abilities. Despite the great theoretical potential, in practice is much harder to determine success with *iML* than with *automatic ML (aML)*, since we are adding a subjective agent, which makes the results harder to replicate. This combination of agents was called HCI-KDD approach (Human–Computer Interaction (HCI) and Knowledge Discovery/Data Mining (KDD) (Holzinger, 2016).

In general terms, AI is being applied daily in the medical field, and many more applica-

tions are constantly being discovered and developed. “AI is poised to revolutionize many aspects of current clinical practice in the foreseeable future. AI systems can enhance clinical decision-making, facilitate disease diagnosis, identify previously unrecognized imaging or genomic patterns associated with patient phenotypes, and assist in surgical interventions for various human diseases (Yu et al., 2018).

1.4. Project Structure and Methodology

This work is structured into the following five general blocks. Each of these may be further divided into different chapters:

1. **Medical and Technological Context:** To help the comprehension of data and its relevance, an introduction to cardiology is included, explaining the fundamentals of the anatomy and physiology involved, the most common heart related conditions and the identified risk factors.

   Following, a small theoretical context of Artificial Intelligence and Machine Learning is included, explaining the basis of AI algorithms, their classification and the accuracy measures they are evaluated upon.

2. **Data Processing and Statistical Analysis:** Small processing and normalisation was required and statistical analysis is included to understand the data used throughout the project.

3. **Development and Results:** After having introduced the necessary information to grasp the concepts, the details of the development and programming are explained. This section is where the algorithms are exposed, configured, trained and tested and the results presented.

4. **Conclusions and Further Studying:** Lastly, a deeper analysis of the results and the techniques are included in the conclusions, followed by some guidelines for

further studying. This chapter comments on the possible related projects, based on observations and ideas obtained during its development (both chapters available in English and Spanish).

5. **Appendices:** The appendices present additional data and code obtained from the development of the project. This information is presented to further analyse methods and resources, although it is not necessary to understand them.

To carry out the project and accomplish the objectives in a timely manner, the following steps were taken in this approximate order. The final report was being written simultaneously, and revised continuously.

1. AI and ML research: selecting the algorithms to be implemented, the libraries and technologies to use, etc.

2. Data search, analysis and processing: finding the correct database, according to characteristics of attributes and number of samples included.

3. Medical background: research into the cardiovascular system, diseases and risk factors, oriented to the information available form the dataset.

4. Programming, configuration and implementation of algorithms: experiment with the selected algorithms and technologies to find the best setting for this project.

5. Results: analysis of the previously obtained results.

6. Conclusions and future research: determine the outcome of the project, the options to improve and other further research lines.

1.5. Resources

Throughout the definition and development of the project, many different resources have been sought to satisfy its needs:

1.5. Resources

- Python 2.7.16: The entire programme has been built on Python, given the available resources to implement AI.
  - **Scikit - learn** offers immense variety of powerful resources for AI. The 5 different classifiers have been taken from this library.
  - **Pandas** is a package for numerical analysis, including statistics, array and matrix treatment, etc.
  - **matplotlib** offers pyplot, an extensive library for plotting graphs and diagrams.
  - **Statistics, Seaborn, Random** are other packages that have aided in the development of the different sections.
- GitHub: Version Control System has been used for tracking versions and changes in files. All the project's files and information can be obtained from the cloud.
- PyCharm - JetBrains: IDE for developing the programme. Includes friendly interfaces for plotting, coding and debugging.
- Google Colaboratory: Used to share code with others and to run the code on Google's available GPU on remote.
- Kaggle: Database repository from which the data used for this project has been obtained.

2. Cardiovascular System

The Circulatory System (CS) is a major apparatus in the body of any mammal. It is in charge of blood transport between the heart, the lungs and all the other organs. The heart is the main organ in this system, acting as the pump that makes it ever-active. To function restlessly, the heart has the capability of creating electric impulses on its own and thus producing muscular contraction. It is of vital importance that the circuit works properly; that the heart is capable of pumping enough blood with the correct pressure, that the blood vessels are capable of keeping such pressure and that the blood carries sufficient nutrients and molecules to the organs.

The pumped blood is transported to and from the organs through arteries and veins, respectively. All organs need blood to provide particular nutrients to function properly, and to take away waste produced during each organ's specific metabolic process. Unfortunately, the body is not capable of distinguishing between molecules that are part of the cycle, such as oxygen or carbon dioxide, from those which are not, like drugs, excess fat or cholesterol, and therefore can also provide organs with toxic substances that will endanger the metabolism.

Additionally, the CS participates in some homeostatic functions, such as regulating blood pressure, body temperature and hormone concentration, adapting the body to diverse states when exercising, wounded, changing position, amongst others (Costanzo, 2018).

2.1. Anatomy and Physiology

To fully understand the CS, it is fundamental to have general knowledge of its anatomy and components. The main two organs are the heart and the vessels that carry the blood (arteries, veins and capillaries).

The heart is anatomically and physiologically divided into two sides: right and left. Each side is further divided into two chambers: atrium (superior) and ventricle (inferior), connected by a one-direction valve that allows blood flow from the atria to the ventricles (and not in the opposite direction). This division allows the two different blood circuits: pulmonary and systemic.

- **The pulmonary circuit** connects the unoxygenated blood from the right side of the heart with the lungs, where gas exchange happens: Oxygen is carried by haemoglobin from the lungs to the capillaries and carbon dioxide does the opposite path to be removed from the body in the next exhalation.

- **The systemic circuit** carries the recently oxygenated blood to the rest of the organs, including the heart and the lungs themselves.

Figure 2.1 shows a graphic representation of the cardiac cycle, which is organised, in general terms, in these 8 steps (Costanzo, 2018):

1. Oxygenated blood fills the left ventricle.
2. Blood is ejected from the left ventricle into the aorta
3. Cardiac output is distributed among various organs.
4. Blood flow from the organs is collected in the veins.
5. Venous return to the right atrium.
6. Mixed venous blood fills the right ventricle.
7. Blood is ejected from the right ventricle into the pulmonary artery.
8. Blood flow from the lungs is returned to the left atrium via the pulmonary vein.

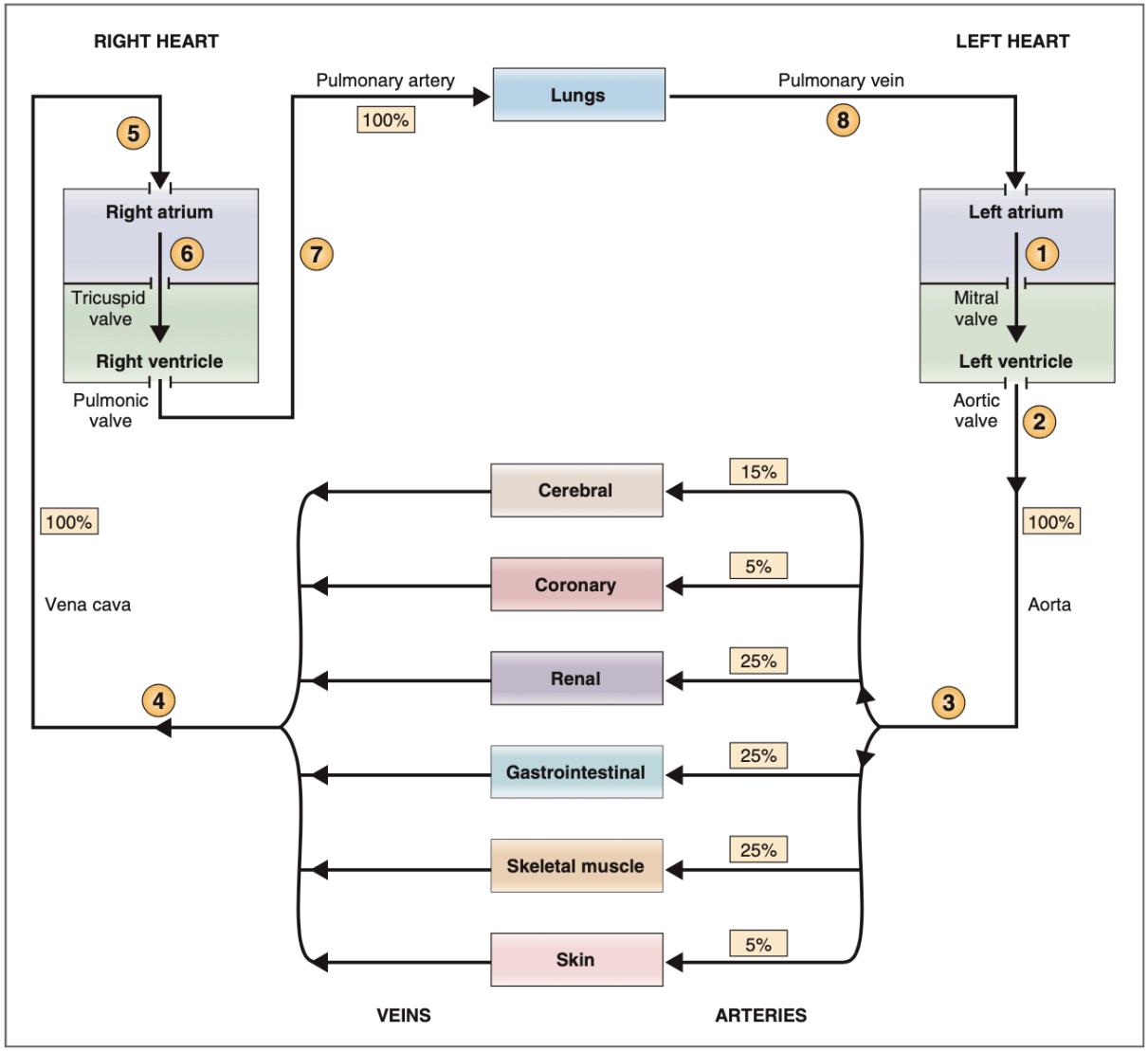


Figure 2.1.: Schematic diagram of the circuitry of the cardiovascular system (Costanzo, 2018)

2.1. Anatomy and Physiology

The heart is an organ composed of cardiac muscle, meaning it can be contracted and expanded. Much like any other organ, veins and arteries provide the heart with blood irrigation (transporting the necessary molecules) to perform all its duties; these are called **coronary arteries and veins**. The heart is the only organ in the body capable of creating electrical stimuli by itself, due to specialised tissue called the sinoatrial (SA) and atrioventricular (AV) nodes. The SA node makes the atria contract, starting the electrical cycle of the heart and thus acting as a pacemaker. When the impulse reaches the AV node, the ventricles will contract, expelling the blood from the heart into the aorta. The nodes are capable of generating 60 to 100 stimuli per minute under normal conditions (The Johns Hopkins University, 2020). Figure 2.2 shows a cross section of the heart, where the four chambers are easily identifiable and the nodes are clearly indicated. The arrows represent the direction of the activation impulse.

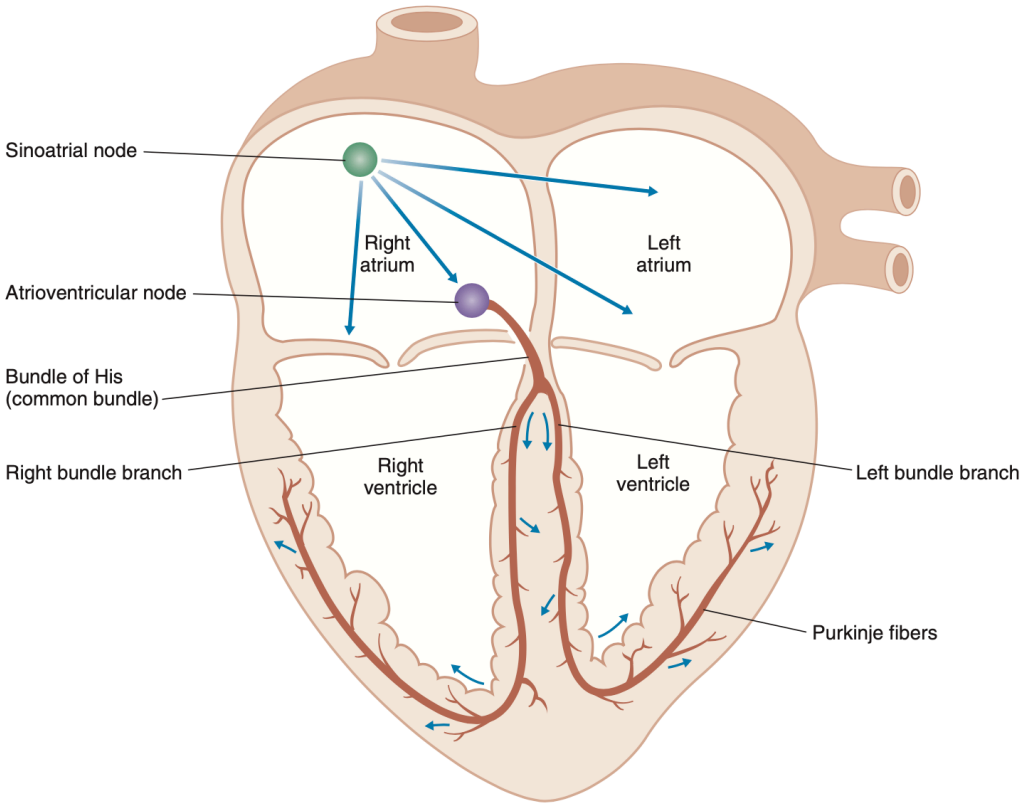


Figure 2.2.: Diagram of the heart and sequence of activation (Costanzo, 2018)

The time between one electrical impulse and the next is called **diastole**, and **systole** is when the heart is contracting to eject the gathered blood. "Diastole represents ventricular filling, and systole represents ventricular contraction/ejection. Systole and diastole occur in both the right and left heart, though with very different pressures" (Pollock, 2019). In other words, **systolic pressure** refers to the highest pressure the blood is exerting to the arterial walls when the heart is beating (normal values are around 120 mmHg) and **diastolic pressure** refers to the lowest pressure on the arteries at rest, namely, in the time between beats (normal values are around 80 mmHg) (American Heart Association, 2017). Figure 2.3 shows two complete heart cycles, and the difference in blood pressure. The rising curve represent ventricular filling, and the decreasing curve represents the arterial pressure after ejection.

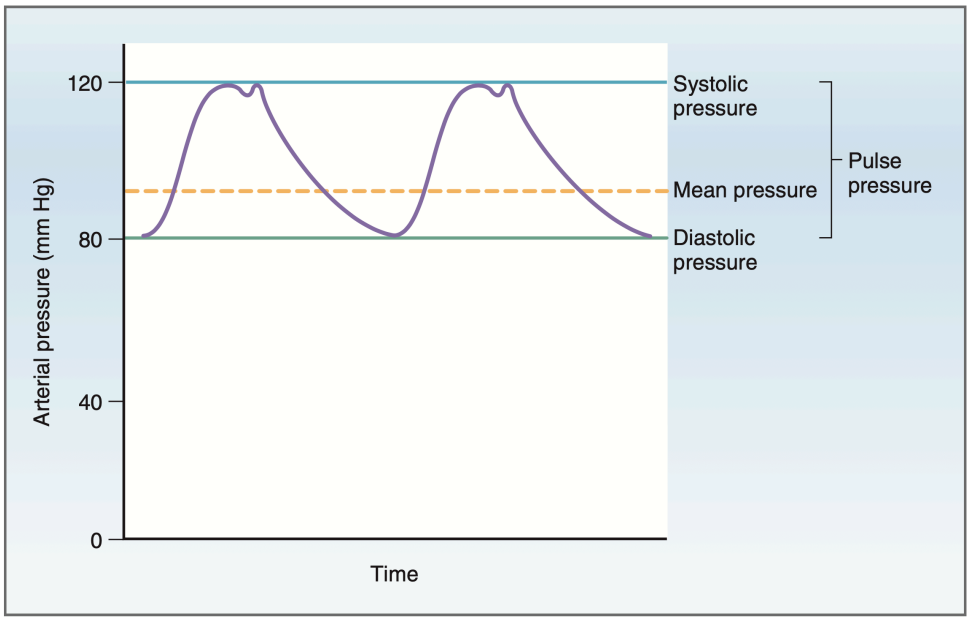


Figure 2.3.: Systemic arterial pressure during the cardiac cycle (Costanzo, 2018)

2.2. Cardiovascular Disease

Cardiovascular Disease (CVD) is a general term that refers to any health condition that affects the heart and the vessels (NHS, 2018). Amongst the wide variety of CVDs, there

are some that are far more frequent than others. These are briefly explained below:

- Atherosclerosis: occurs when the diameter of the arteries is decreased by plaque deposits, making them stiffer and less compliant. As a result, systolic and diastolic pressure will both be increased (Costanzo, 2018). Figure 2.4 shows a graphic comparison of a healthy artery and a partially clogged one.
- Coronary Heart Disease (CHD): heart's blood supply is blocked or interrupted by a build-up of fatty substances in the coronary arteries, namely atherosclerosis affecting the coronary arteries (heart blood supply itself) (NHS, 2020).
- Aortic Aneurysms: occurs when the aorta develops a tear in the internal layer. It is extremely dangerous as it can cause death very few minutes after its rupture. It is caused by high blood pressure and atherosclerosis, amongst other conditions (Mayo Foundation for Medical Education and Research, 2019a).
- Cardiomyopathy: it is a spectrum of cardiac dysfunction, meaning the heart cannot pump the necessary volumes of blood, due to diverse reasons. Causes include prolonged high blood pressure, alcohol and drug abuse and metabolic disorders (obesity, diabetes, etc.) (Mayo Foundation for Medical Education and Research, 2019b).

2.2.1. Risk Factors

As mentioned before, there are a number of factors that contribute widely to the presence and severity of CVDs. It is important to note that many of these factors are related, and are frequently presented together (like obesity and high cholesterol, for example), complicating the study of one single risk factor. However, there is evidence that, alone or combined, all these factors contribute to a higher risk of suffering from CVDs:

1. **Obesity:** The relationship between weight and heart disease does not come as surprise, as an obese person will have more fat and thus more chance of suffering

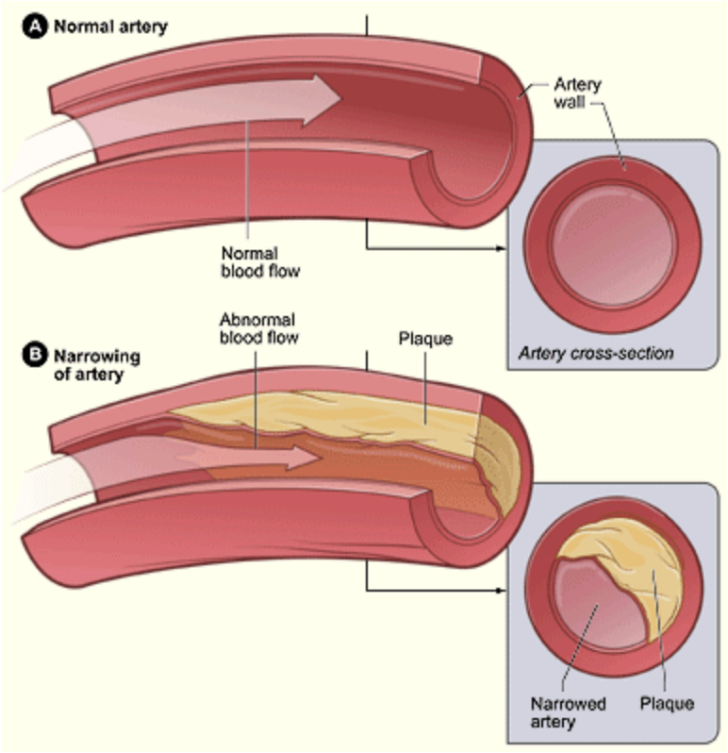


Figure 2.4.: Cross section of an artery with plaque buildup (NHLBI, 2020)

from CVDs. "Irrespective of metabolic health, overweight and obese people have higher coronary heart disease risk than lean people" (Lind, Risérus, & Ärnlöv, 2020).

2. **Physical activity:** Constant exercise and a healthy lifestyle can, in general, impact very positively to the treatment of diseases: prevent or delay the onset of type 2 diabetes, reduce blood pressure and help reduce the risk for heart attack and stroke (American Heart Association, 2015).

3. **Cholesterol Levels:** Cholesterol builds up is one of the main causes of atherosclerosis. It has been consistently demonstrated that higher long-term LDL cholesterol and non-high-density lipoprotein cholesterol concentrations are associated with increased CVD risk (Gidding & Allen, 2019).

4. **Glucose / Diabetes:** Diabetes is not only an alteration to the blood sugar levels, but it affects the system generally. Studies report a positive association between

hypertension and insulin resistance (American Heart Association, 2015).

5. **Smoking:** There is evidence that smoking causes about 1 in 10 deaths from CVDs. Tobacco smoke contributes to CVDs as it increases atherosclerotic plaque and the chance of thrombosis (U.S. Department of Health and Human Services, 2010).

6. **Alcohol intake:** It has been studied that alcohol intake can in fact have a positive effect on the metabolism. However, when large volumes of alcohol are consumed, it increase the risk of disease, particularly, CVDs. There is also evidence that alcohol damages the heart muscle, causing alcoholic cardiomyopathy (Piano, 2017).

3. Artificial Intelligence

3.1. History and Fundamentals

Artificial Intelligence (AI) was born as a discipline in the mid 20th century, but it was only towards the end of the century that started to gain more attention, mainly due to the progress in computer technologies and data availability (Haenlein & Kaplan, 2019). It can be defined as "a science of finding theories and methodologies that can help machines understand the world and accordingly react to situations the same way humans do" (Joshi, 2017). Machine Learning (ML) is a subset of the wider umbrella of AI, and can be defined as "a collection of algorithms and techniques used to create computational systems that learn from data in order to make predictions and inferences" (Manohar, 2017). Figure 3.1 shows a very simple cycle of how these algorithms may learn from the data.

The concepts of AI can be traced back to Alan Turing, as he began to question the intelligence of machines, and developed the widely used *Turing Test*, to distinguish machine from human intelligence (Haenlein & Kaplan, 2019). There have been many stages in the history of AI and ML before it could reach the extent we know today.

Due to its potential and exponential growth, new applications are constantly being developed. Figure 3.2 shows just a few of the many domains in which AI is being applied.

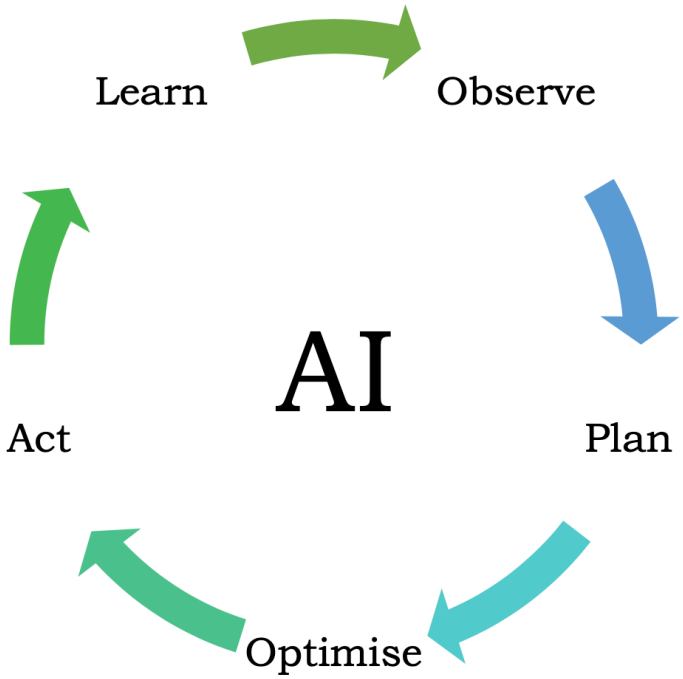


Figure 3.1.: Intelligent algorithms cycle

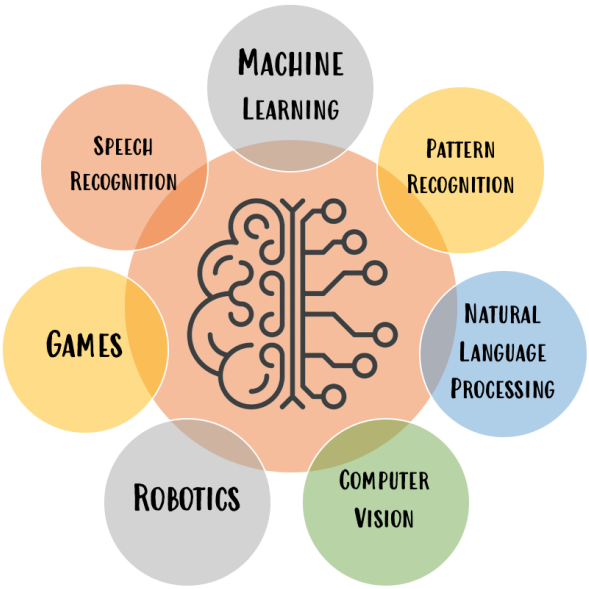


Figure 3.2.: Examples of AI applications

3.2. Supervised and Unsupervised ML

It has been previously mentioned how algorithms learn from their mistakes or success. This can only be done when talking about Supervised Learning; this is when the expected output is known, so the algorithm is either right or wrong based on labelled data. On the other hand, Unsupervised Learning is the applications of algorithms on untagged data, meaning all the information must be inferred by the algorithm itself. This later one is more commonly used when finding patterns or identifying objects, rather than predicting.

Supervised Machine Learning is mainly used for regression and classification of data. Regression is useful when predicting the outcome of a certain action in the future, based on available results. Depending on the accuracy and the shape of the slope, regression can be **linear** or **polynomial**. Classification algorithms are capable of determining the class of a sample, grouping it by the outcome of predictions. For example, a classifier would determine if a certain sample belongs to class a or b (in which case it would be a binary classifier) or predict the class from a wider range (in which case it is called a multi-class classifier).

Unsupervised Learning can be used for anomaly or pattern detection and dimension reduction, amongst many other applications. "Unsupervised learning finds applications in diverse fields of study, including market segmentation, stock markets, natural language processing, computer vision, and so on" (Joshi, 2017).

3.3. Performance Measures

When working with supervised algorithms, there need to be measure standards to compare the results in a objective manner. These metrics show how precise, reliable or sensitive the chosen algorithm is. They are based on the comparison between the obtained and the expected outcomes:

- **True Positive (TP):** The expected and the predicted outcome were both *positive*.

- **True Negative (TN):** The expected and the predicted outcome were both *negative*.
- **False Positive (FP):** The algorithm predicted *positive* and the expected outcome was *negative*.
- **False Negative (FN):** The algorithm predicted *negative* and the expected outcome was *positive*.

An easy way to show the results of any binary classification algorithm is to use a Confusion Matrix, shown in Table 3.1.

		PREDICTED	
		FALSE	TRUE
ACTUAL	FALSE	TN	FP
	TRUE	FN	TP

Table 3.1.: Example of Confusion Matrix

Based on these four values, the most commonly used performance measures are:

1. Accuracy: Indicates the ratio of prediction that were correct.

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

2. Misclassification Rate:

$$\frac{FP + FN}{TP + TN + FP + FN} \tag{3.2}$$

3. True Positive Rate (TPR) / Recall / Sensitivity:

$$\frac{TP}{TP + FN} \tag{3.3}$$

4. Specificity:

$$\frac{TN}{TN + FP} \tag{3.4}$$

5. Precision:

$$\frac{TP}{TP + FP} \tag{3.5}$$

Cross Validation

To work with supervised algorithms, data must be split into different sets. This means the algorithms learns from one set and it is then evaluated on another one. Using the same data for both processes would result in **overfitting**; the algorithm would only be repeating the exact values it has recently learnt, and its ability to predict would not be tested. **K-fold** means dividing the available data into K different sets, training using K-1 sets and testing on the remaining one. Figure 3.3 shows a representation of a 6-fold cross validation.

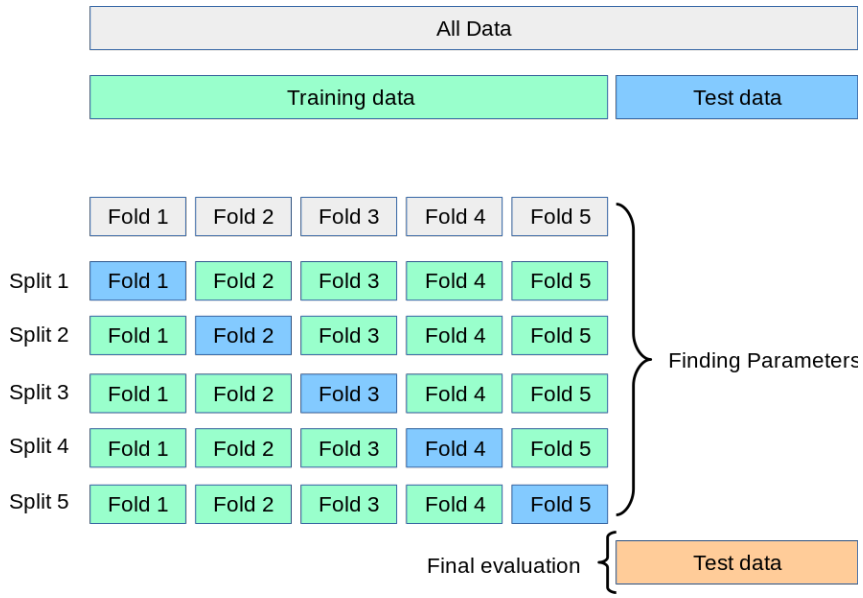


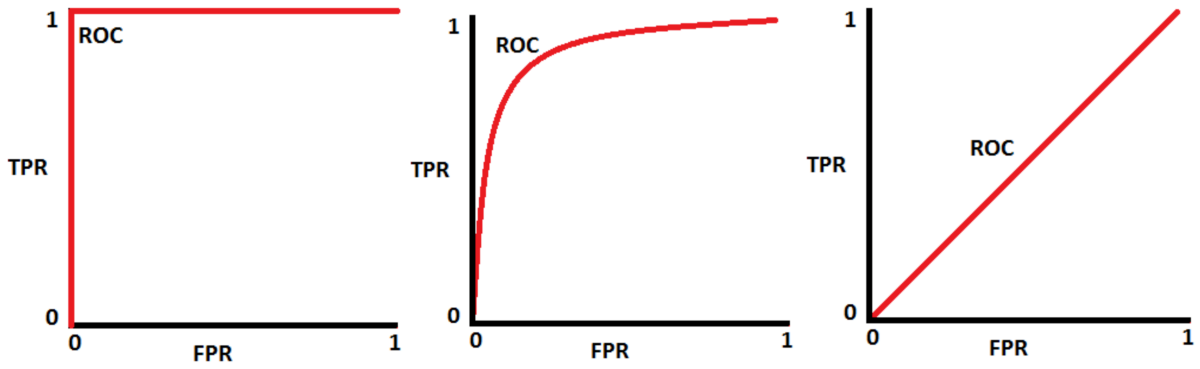
Figure 3.3.: 6-fold cross validation structure (Pedregosa et al., 2011)

ROC Curve

Another way to show results for a binary classification algorithm is to print a Receiver Operating Characteristics (ROC) curve. The abscissa axis shows False Positive Rate (FPR) (equation 3.6) and ordinate axis represents True Positive Rate (TPR) (equation 3.3). The Area Under Curve (AUC) shows how good the model is when separating between classes, thus higher values mean better performance.

$$FPR = 1 - specificity = \frac{FP}{TN + FP} \tag{3.6}$$

Figure 3.4 shows three example of ROC curves, which represent the best possible outcome (figure 3.4a), a more realistic good result (3.4b) and the worst possible curve (3.4c). In the first case, the model is capable of predicting correctly 100% of cases, where in the last, it is not capable of predicting correctly at all. The middle graph (3.4b) represents the most frequent situation for good results; above the diagonal and a steep curve.



(a) best (b) good (frequent) (c) worst

Figure 3.4.: Significant ROC curves (Narkhede, 2019)

3.4. Artificial Intelligence in Healthcare

As early as the concepts of Artificial Intelligence (AI) were born, medicine has been identified as an extremely promising field for application, aiding diagnosticians to make better

decisions, detect anomalies much more efficiently than humans, bring better healthcare to remote areas, or reduce staff strain. These are just a few fields where AI has shown rich potential. The large-scale annotated clinical data, availability of open-source ML packages and many other factors have fuelled the recent exponential growth in AI (Yu et al., 2018).

Figure 3.5 shows the general workflow of a decision support systems, helping clinicians to determine the state of a patient, treatment or any other decision needed.

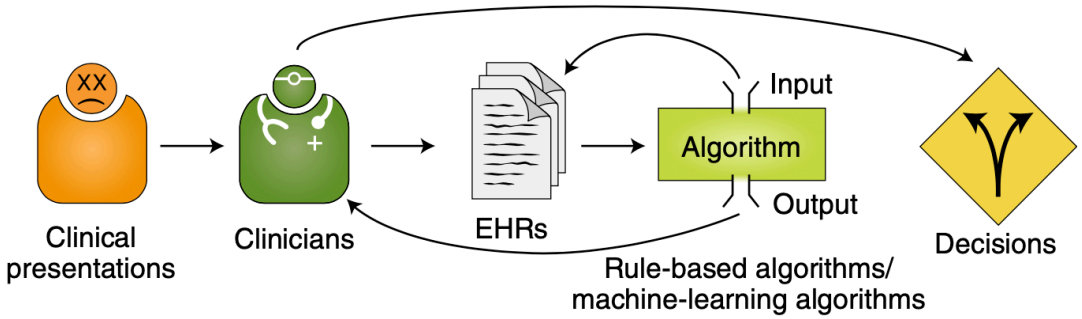


Figure 3.5.: Integrative decision support systems (Yu, Beam, & Kohane, 2018)

AI and more particularly ML algorithms need large amounts of data to achieve good results. This is one of the limitations when applying them to healthcare, since there is still great fraction of on paper medical data. It is only recently that most hospital have adopted EHRs. As of 2017, only nearly 60% of countries in the EU claimed to have EHRs (WHO, 2017). There is also a need for integration between the many information sources: "(...) complex patient related data sets, resulting from various sources including picture archiving and communication systems (PACS) and radiological information systems (RIS), hospital information systems (HIS), laboratory information systems (LIS), physiological and clinical data repositories, and all sorts of -omics data from laboratories, using samples from Biobanks" (Holzinger, 2016).

4. Data Structure and Analysis

Since the focus of this project is to predict the values rather than processing the data, these steps have been minimised to drift attention to the creation and implementation of the ML algorithms. The source of the data used is from a public database (https://www.kaggle.com/sulianova/cardiovascular-disease-dataset/data#) and the dataset had already undergone some pre-processing, simplifying those steps.

4.1. Data Structure

The raw database downloaded contains observation of 70000 patients, each of them with 13 columns. Table 4.1 shows all attributes for each observation, and some relevant information about datatype and possible values.

The values are classified in 3 different groups (excluding the target value), depending on their nature: *Objective Features* refer to personal data, such as age, biological gender, height and weight. Each is measured with an appropriate unit (days, binary, cm and kg, respectively). The second group of attributes are the *Examination Features*, including values obtained from a physical and physiological exam, such as blood pressure (high and low), cholesterol and glucose levels. The last two are obtained in a standard blood test, and have been classified into 3 groups (normal, above normal and well above normal). Lastly, the *Subjective Features* include information about the lifestyle of the patient, cannot be validated and thus are variable due to subjective perception. These include smoking, alcohol consumption and physical activity, and are all measured as binary values.

	attribute	observation type	name	data type
0	id	identifier	id	int
1	Age	Objective Feature	age	int (days)
2	Gender	Objective Feature	gender	categorical code
3	Height	Objective Feature	height	int (cm)
4	Weight	Objective Feature	weight	int (kg)
5	Systolic Blood Pressure	Examination Feature	ap_hi	int (mmHg)
6	Diastolic Blood Pressure	Examination Feature	ap_lo	int (mmHg)
7	Cholesterol	Examination Feature	cholesterol	1: normal, 2: above normal, 3: well above normal
8	Glucose	Examination Feature	gluc	1: normal, 2: above normal, 3: well above normal
9	Smoking	Subjective Feature	smoke	binary
10	Alcohol Intake	Subjective Feature	alco	binary
11	Physical Activity	Subjective Feature	active	binary
12	Cardiovascular Disease	Target Variable	target	binary

Table 4.1.: Attribute names, types and values.

4.2. Statistical Analysis

Although the data had already been pre-processed, some attributes presented outliers, and thus had to normalised to improve the accuracy of the methods. This has particularly been the case of systolic and diastolic blood pressure (ap_hi and ap_lo).

Table 4.2 shows the initial statistical information for both of these attributes. The highlighted values helped identified the outliers. As it can be seen, these two columns present values outspread in an extremely wide range, also seen by the standard deviation. To correct this values, these two columns have been normalised, removing entries that fell below the 2.5% or above 97.5%, verifying that negatives values have been removed, due to their physiological significance. This has been done using the **DataFrame** functions *drop()* and *quantile()*, available in Python's Pandas package.

Table 4.3 shows the data after the removal of the outliers. As it can be seen, now these attributes fall within the expected range (from information shown in Figure 2.3). The

number of samples has dropped from 70000 to 66193, which means that **3807** samples had insignificant values that could be harmful to the algorithm.

With the already normalised data, we can further analyse the relation between the variables. Python's package *Seaborn* allows to easily build a correlation map, shown in figure 4.1.

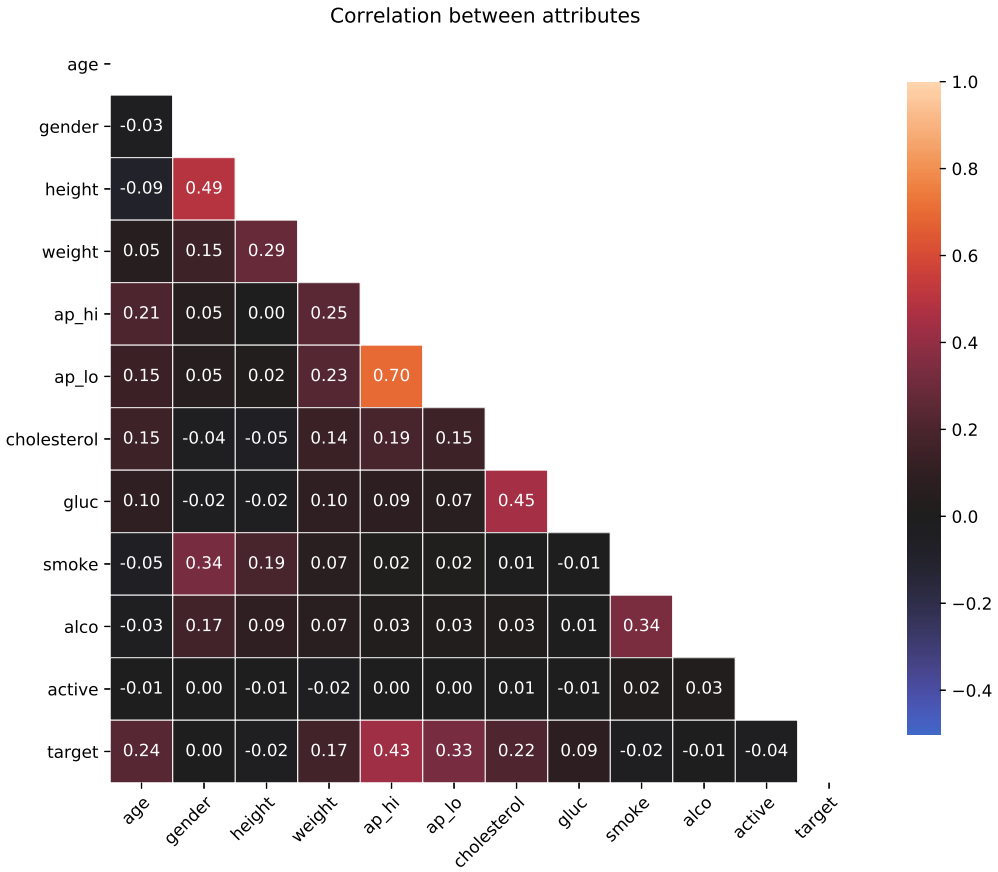


Figure 4.1.: Variable correlation heatmap

	age	gender	height	weight	ap_hi	ap_lo	choles	gluc	smoke	alco	active	target
count	70000	70000	70000	70000	70000	70000	70000	70000	70000	70000	70000	70000
mean	19467.7	1.3	164.3	74.2	128.9	96.6	1.4	1.2	0.1	0.1	0.8	0.5
std	2467.2	0.5	8.2	14.4	158.9	187.6	0.7	0.6	0.3	0.2	0.4	0.5
min	10798.0	1.0	55.0	10.0	-150.0	-70.0	1.0	1.0	0.0	0.0	0.0	0.0
25%	17663.0	1.0	159.0	65.0	120.0	80.0	1.0	1.0	0.0	0.0	1.0	0.0
50%	19703.5	1.0	165.0	72.0	120.0	80.0	1.0	1.0	0.0	0.0	1.0	1.0
75%	21324.0	2.0	170.0	82.0	140.0	90.0	2.0	1.0	0.0	0.0	1.0	1.0
max	23713.0	2.0	250.0	200.0	16020.0	11000.0	3.0	3.0	1.0	1.0	1.0	1.0

Table 4.2.: Statistical measures before normalisation

	age	gender	height	weight	ap_hi	ap_lo	cholest	gluc	smoke	alco	active	target
count	66193	66193	66193	66193	66193	66193	66193	66193	66193	66193	66193	66193
mean	19473.22	1.35	164.42	74.12	126.23	81.17	1.36	1.22	0.09	0.05	0.80	0.49
std	2465.10	0.48	8.15	14.13	14.46	8.40	0.68	0.57	0.28	0.22	0.40	0.50
min	10798.00	1.00	55.00	11.00	100.00	60.00	1.00	1.00	0.00	0.00	0.00	0.00
25%	17676.00	1.00	159.00	65.00	120.00	80.00	1.00	1.00	0.00	0.00	1.00	0.00
50%	19708.00	1.00	165.00	72.00	120.00	80.00	1.00	1.00	0.00	0.00	1.00	0.00
75%	21329.00	2.00	170.00	82.00	140.00	90.00	1.00	1.00	0.00	0.00	1.00	1.00
max	23713.00	2.00	250.00	200.00	170.00	100.00	3.00	3.00	1.00	1.00	1.00	1.00

Table 4.3.: Statistical measures after normalisation

As it was expected, the attributes that are most related to one another are the two blood pressures, systolic and diastolic, with the highest correlation value of **0.7**. There are other attributes that show higher correlation indices, such as gender and height, or weight and height. These correlations are not significant to our analysis, since the difference in height is known to be common for the different genders. Similarly, the correlation between height and weight is not valuable, as it is known that they are heavily linked.

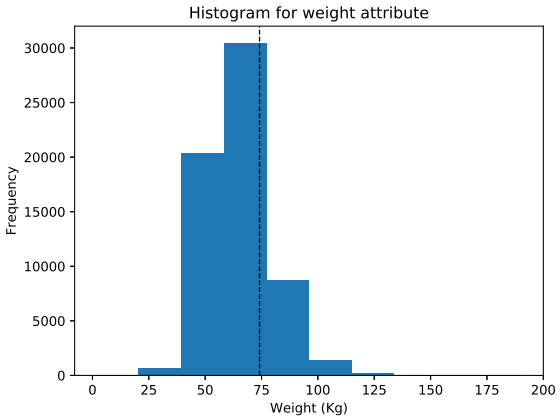
There are however, some correlations that are of special value to us, and are those involving the target value. This initial analysis gives a preview of which could be the variables that may be most related to the presence of the disease, which is also demonstrated by the theory explained in chapter 2. Here are the four variables that have the highest correlation with the *target* attribute:

- ap_hi - target => 0.43
- ap_lo - target => 0.33
- age - target => 0.24
- cholesterol - target => 0.22

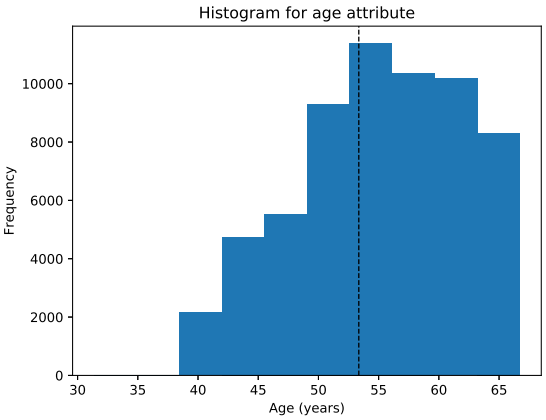
Figure 4.2 show the frequency for all continuous attributes in the dataset. When it seemed significant, the sample mean was added to the histogram for reference. Figures 4.2c and 4.2d have been added additional information to show the normal values for these measurements.

Table 4.4 shows the frequencies of the discrete attributes not included in Figure 4.2, for a total of 66193 samples.

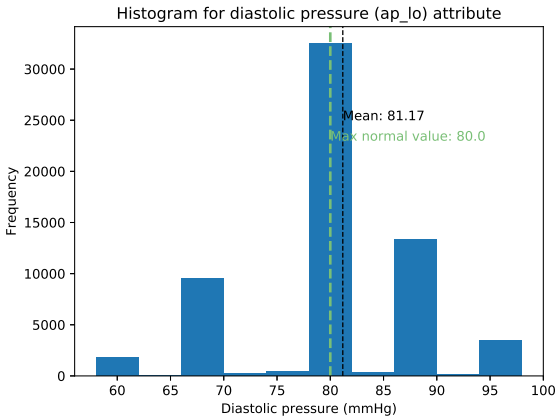
With an overview of the data, it can be safely assumed that most of the individuals sampled are *healthy*, meaning they have average weight, do some physical activity, do not ingest high volumes of alcohol, normal levels of cholesterol and glucose, do not smoke, and their blood pressure is around the safe values.



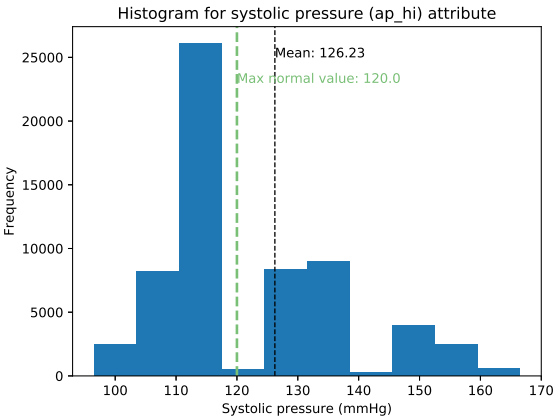
(a) weight



(b) age



(c) diastolic pressure



(d) systolic pressure

Figure 4.2.: Frequency for data attributes

Gender	65.0% (female)	35.0% (male)	-
Cholesterol	75.3% (normal)	13.3% (above normal)	11.4% (well above normal)
Glucose	85.2% (normal)	7.2% (above normal)	7.6% (well above normal)
Smoking	91.3% (no)	8.7% (yes)	-
Alcohol Intake	94.7% (no)	5.3% (yes)	-
Physical Activity	19.7% (no)	80.3% (yes)	-
Cardiovascular Disease (target)	50.7% (no)	49.3% (yes)	-

Table 4.4.: Frequency of discrete attributes

5. Implementation of Algorithms

For the programme itself, five different algorithms have been selected. There is an immense range of possibilities to choose from, and prior to the final selection, a small analysis of the most common classifiers was done, to select the ones that fitted the designed programme better. The five algorithms presented below were found to be the most commonly used and the easiest to implement and understand:

1. **Decision Tree:** Creates a set of rules from the data features that finally lead to a classification.

2. **Random Forest:** Ensemble of many single decision trees. The final output will be the most frequent prediction from the single trees.

3. **Support Vector Machines:** Divides the data through different planes and thus classifies the samples according to the *area* they have fallen into.

4. **K-Nearest Neighbour:** Classifies an instance according to the most frequent class of the K nearest neighbours (K being a predefined integer).

5. **Multilayer Perceptron:** Is a type of Neural Network, where the model may have as many hidden layers as needed, aside from the input and output. It can learn non-linear functions (Pedregosa et al., 2011).

As explained before, Python has an immense amount of resources already implemented and ready to use. Listing 5.1 shows the general approach for these algorithms, using the package *sci-kit learn*.

The outputs obtained from each algorithm are the ROC curve, the AUC and the confusion matrix. Each classifier was run 5 times, to check consistency of results, which are also presented in a table, including the mean accuracy and the standard deviation for all the runs. For those iterations, the confusion matrix and ROC curve shown are from the run that provided the best results for the test set.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from matplotlib import cm, pyplot as plt
from sklearn import neighbors, datasets, metrics, svm, neural_network,
    ensemble, tree

# select size of sample to be taken (about 20000 entries)
RANDOM_SAMPLE_FRAC = 0.3

if __name__ == "__main__":
    data = pd.read_csv('PATH_TO_FILE/FILE_NAME.csv')
    dataframe = data.sample(frac=RANDOM_SAMPLE_FRAC)
    X = np.array(dataframe.drop(['target'], 1))
    y = dataframe['target'].values

    # one third of the sample is for testing (approx. 7000 entries)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.3)

    # This line has been changed according to each classifier.
    clf = callToClassifier()

    clf.fit(X_train, y_train)

    train_prediction = clf.predict(X_train)
    test_prediction = clf.predict(X_test)
```

```
26    print_roc_curve(y_test, X_test, y_train, X_train, clf)
27    print_metrics(y_test, test_prediction)
```

Listing 5.1: General configuration for all supervised algorithms

5.1. Decision Tree

This classifier evaluates information gain on every attribute, based on the reduction of entropy. The attribute that offers the highest information gain will be the next node. Then, the topmost in the tree will represent the attribute that provides the most information of all and each branch represents a decision. Figure 5.1 shows a simple example of a decision tree.

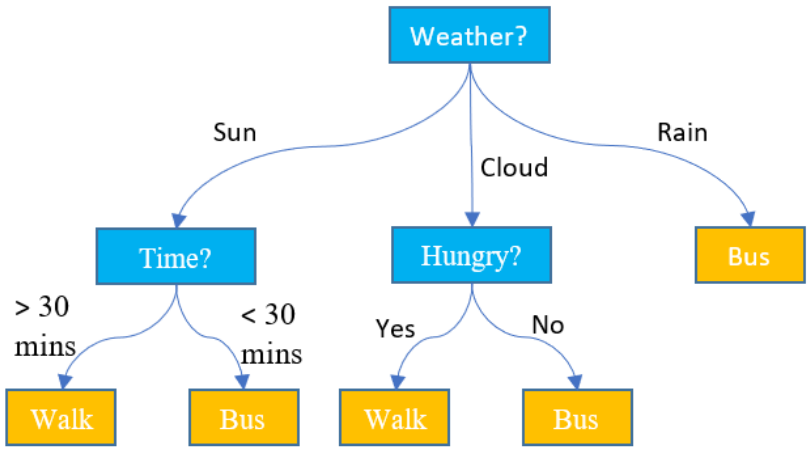


Figure 5.1.: Example of Decision Tree (Hoare, 2020)

Library *tree* from *sci-kit learn* allows to implement decision trees very easily, using the class sklearn.tree.DecisionTreeClassifier(). The attributes of this function can be seen in Table A.1.

5.2. Random Forest

Random forest, as the very name indicates, is an ensemble of individual decision trees where the most predicted outcome will be the ultimate prediction of the forest. This classifier then takes the same principles of DT, and the ensamble is more powerful than a single tree. "A large number of relatively uncorrelated models (trees) operating as a committee, will outperform any of the individual constituent models" (Yiu, 2019).

The reason behind this synergy lies on the uncorrelated results, meaning each tree protects all others from mistakes and biased results. To ensure the trees of the forest are uncorrelated, *bagging* is applied, meaning each tree takes random samples of data from our dataset, and, since the trees are very sensitive to the data they are trained on, the possibility of two trees being identical is very slim. Figure 5.2 shows a very simple example of how a random forest (in this case with 3 estimators), predicts an outcome.

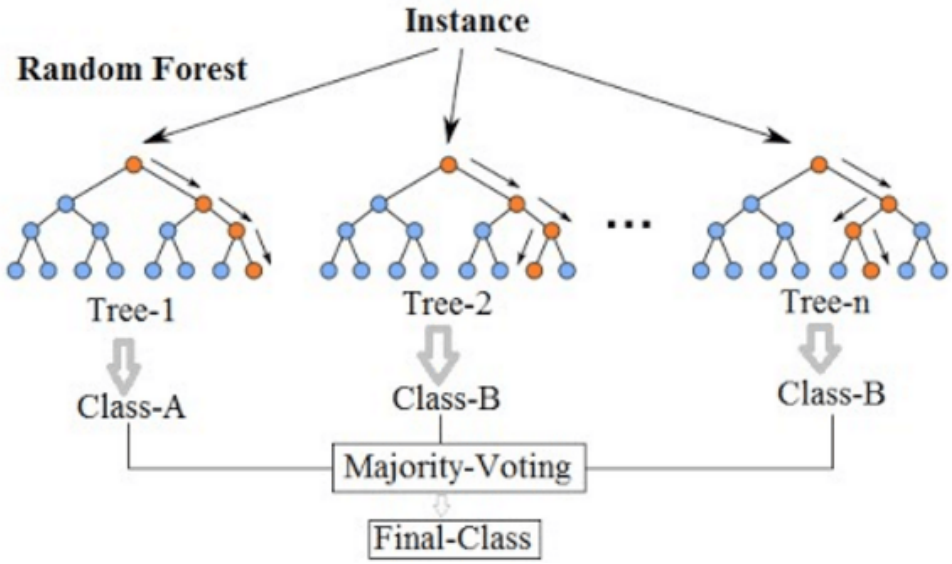


Figure 5.2.: Example of Random Forest (Chakure, 2020)

For the default configuration, the classifier has been initialised with no additional parameters (`sklearn.ensemble.RandomForestClassifier()`). The default implementation of this algorithm comprises the parameters shown in table A.2.

5.3. Support Vector Machines

Support Vector Machine (SVM) classifies data by drawing a hyper-plane of N-1 dimensions (being N the dimensions of the space) between the classes, creating a clear division between the outputs. This way, when evaluating, depending on the position of the outcome on the N-dimensional graph, the sample would be classified into different groups. The objective of SVM is that the hyper-plane is as far away as possible from all the observations, thus finding the largest margin. The algorithm first draws the hyper-plane randomly, and then check the two closest observations, from every class.

Figure 5.3 shows a graphic representation of a 2 dimensional plane. The reality is, however, that most of the times data is not two dimensional and cannot easily be separated by a straight line, as shown in the example. Then kernels act as complementary functions to determine the position of the dividing hyper-plane when it is not linear. These kernels can be linear, polynomials, sigmoid and radial basis function (RBF) (Pedregosa et al., 2011). All the parameters this classifier accepts, their value types and the default are shown in table A.3.

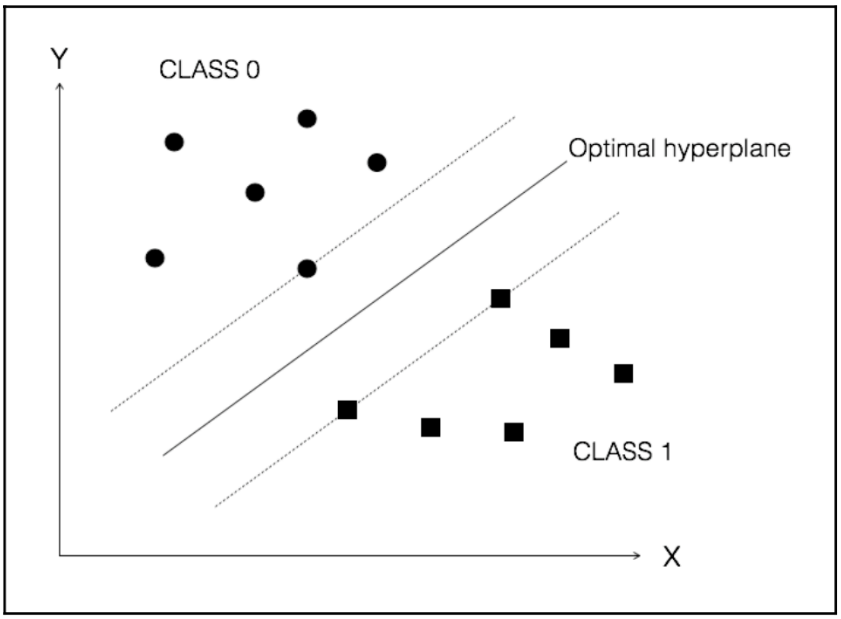


Figure 5.3.: Example of Two-Dimensional SVM (Joshi, 2017)

5.4. K Nearest Neighbours

K Nearest Neighbours (KNN) bases its predictions on the most frequent class of the K nearest points to the one being tested. Different methods can be used to calculate the distance between two points. In this case, the euclidean distance will be used, as it is the most common, and it is the default value for this classifier. Equation 5.1 shows how this distance is calculated for a bi-dimensional problem, where *x1, x2* are the columns of the dataset and i is the index of a specific row.

$$distance = \sqrt{\sum_{n=1}^{N}(x1_i - x2_i)^2} \tag{5.1}$$

Figure 5.4 shows a two dimensional example of KNN algorithm. Note the importance of the value of K; if a small value is chosen, the class of the point would be *Class B*, whilst if a higher value is chosen, the class would be *A*. These principles are directly applicable for higher dimensions.

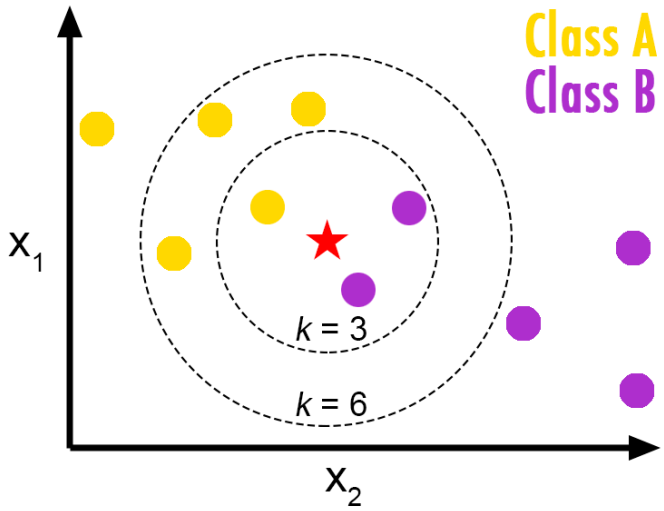


Figure 5.4.: Example of Two-Dimensional KNN

As with the other classifiers, sklearn provides a simple form to construct a KNN Classifier. The only need is to call the constructor: `sklearn.neighbors.KNeighborsClassifier()`.

5.5. Artificial Neural Networks

Multilayer Perceptron (MLP) is a type of Neural Network that allows classification for datasets that are not linearly separable. It consists on several layers that are interwired; figure 5.5 shows a simplified example of a MLP with only one hidden layer. The perceptron can have four different types of activation functions (Pedregosa et al., 2011):

- 'identity': returns f(x) = x
- 'logistic': returns f(x) = 1 / (1 + exp(-x)).
- 'tanh': returns f(x) = tanh(x).
- 'relu': returns f(x) = max(0, x)

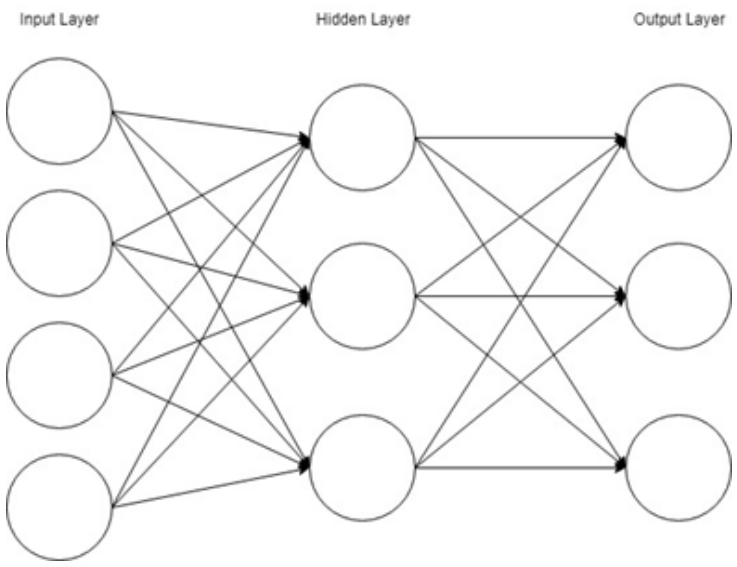


Figure 5.5.: Example of Multilayer Perceptron (DeepAI, 2019)

To implement MLP with sklearn, it was only necessary to initialise the classifier with `sklearn.neural_network.MLPClassifier()` and no additional parameters, for default configuration. All the parameters are show in table A.5.

6. Genetic Algorithms

6.1. Fundamentals of GA

Genetic Algorithms (GA) are based on the same principles as the Natural Selection process proposed by Charles Darwin in 1859 on his work "On the Origin of Species": The strongest individuals are picked out from a random selection of the population: the next generation is created by combination of the genes that have been determined to make the individual stronger. This way, the next generation will have even *stronger* genes, making them more likely to survive. The process is repeated indefinitely throughout generations (Than, 2018). During this process, **selection, crossover, mating** and **mutation** are ways of creating the new, stronger, fitter group, and a **evaluation function** is used to measure how fit the individual is.

In Machine Learning, the process followed by GA is highly similar to that described before; the algorithm will randomly change some characteristics (equivalent to genes) of the individual -making them weaker or stronger (mutation and crossover)-, create new individuals form the combination of others (mating) and a selection (through the evaluation function) will be carried out on each iteration to select which individuals will continue in the sample, and which will be terminated. Since GA are based on a **random** initial selection of individuals, the repetition of the algorithm does not assure identical results. Figure 6.1 shows a general view of how genetic algorithms work.

Figure 6.2 shows a much more detailed workflow of genetic algorithms. It is important to note that there are many ways to carry out the evolution. This methodology shows

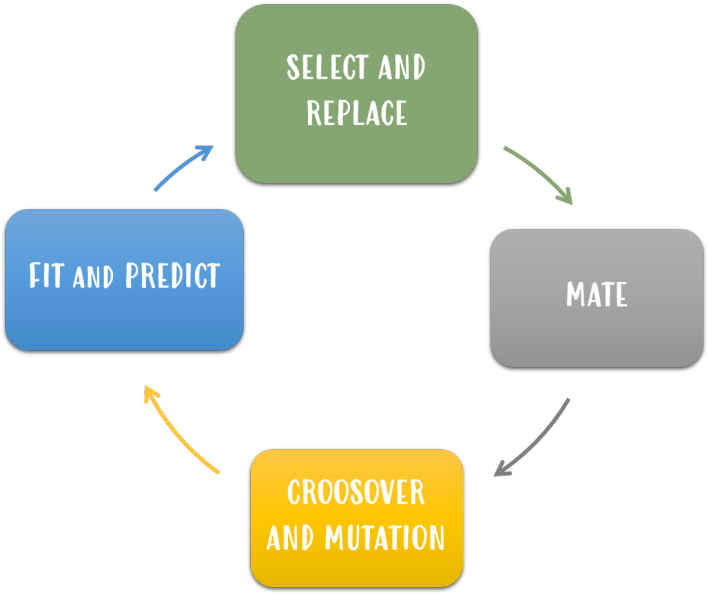


Figure 6.1.: Evolution cycle for classifiers in GA

the one that showed better results for the type of problem proposed.

6.2. Structure and Implementation

To implement the GA dynamics into a Python programme, several classes have been created, each with its own specific parameters. Figure 6.3 shows the classes and their dependencies. The full code for the classes can be found in appendix B.

For the correct functioning of the algorithm, three different datasets are needed: **train, validate and test**, which will represent 40%, 30% and 30% of the total sample, respectively. The training and testing set are functionally equivalent of the ones used in chapter 5, only vary in size. Since the same data for validation and testing cannot be used (to avoid overfitting), the validation set is included in this section to evaluate each algorithm against the evaluation function.

Class **Main** is the orchestrator; sets the initial configuration, imports the data from the csv file and calls all the necessary functions for evolution to occur as shown in figure 6.2.

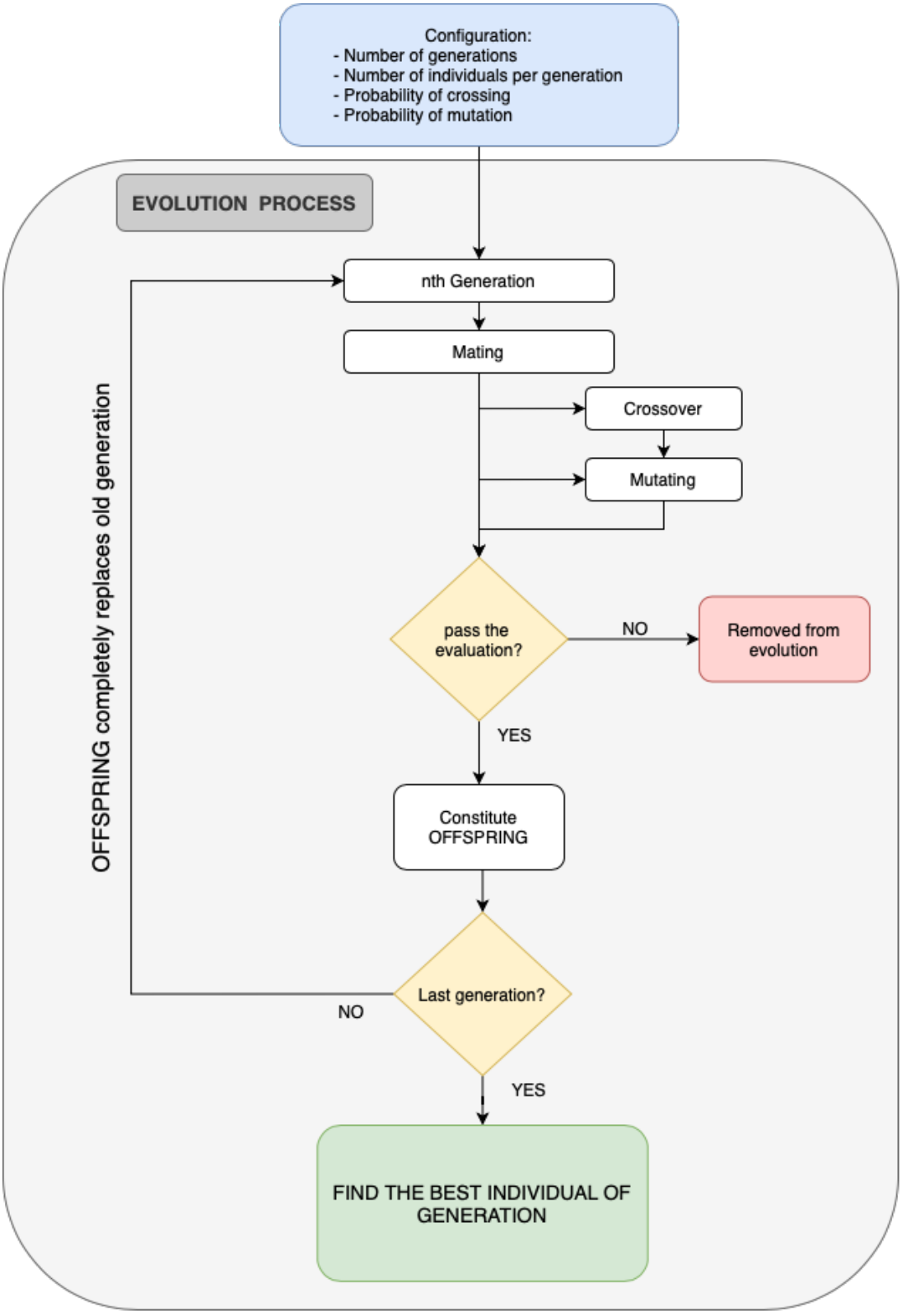


Figure 6.2.: General scheme of a GA

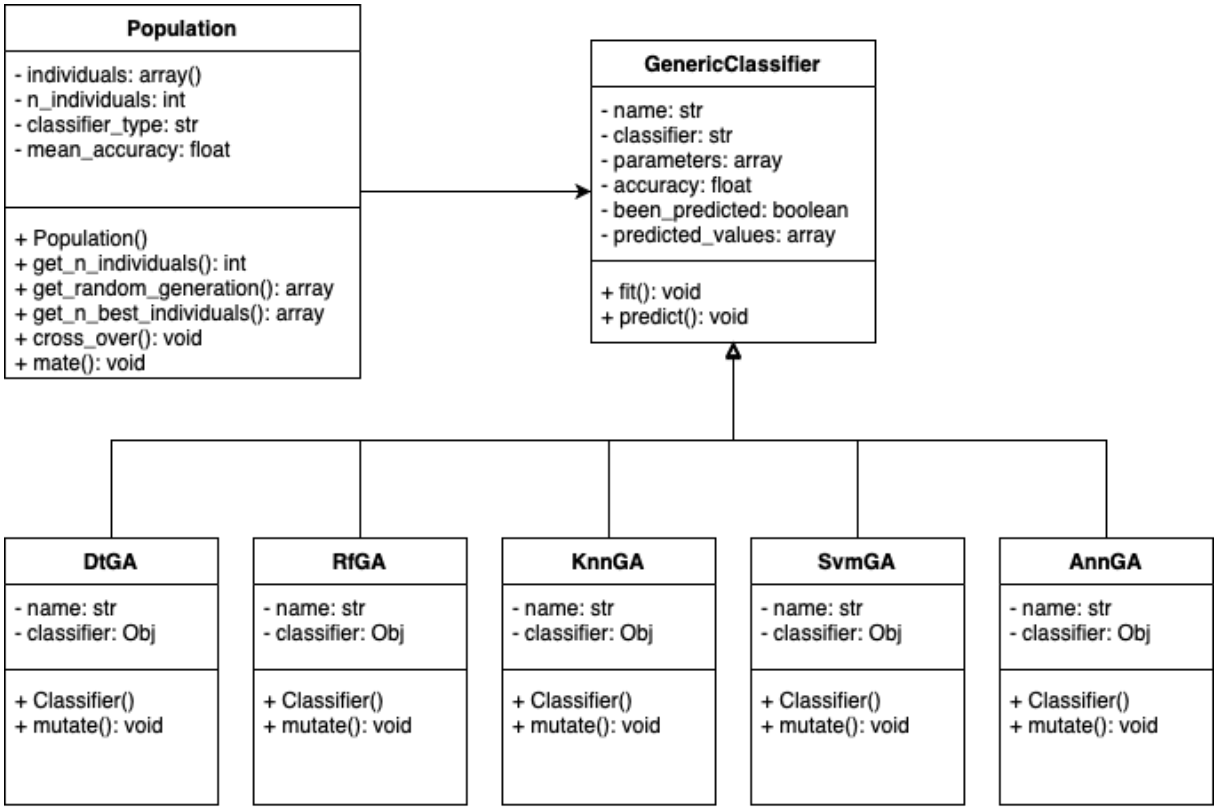


Figure 6.3.: Class diagram for GA programme

Class **Population** represents a vector of individuals of a particular classifier type, hence will be in charge of performing all logic related to the community, including selecting best individuals, cloning, crossing, mating and performing statistical calculations.

Class **Generic Classifier** includes all the attributes, methods and functions common to all classifiers. Thus, all the classifiers will inherit from GenericClassifier.py and only specific methods -such as mutating- are added to each particular classifier. The same algorithms that were implemented in chapter 5 were adapted to fit the logic of GA, all inheriting GenericClassifier.py: RF, DT, KNN, SVM and MLP.

Some values were set initially, and are not part of the optimisation themselves. These values are:

- Number of generations
- Number of individuals on each generation

- Probability of crossing
- Probability of mutating

To better control the changes performed while mutating, crossing-over and mating, not all the available parameters were considered in evolution. This means that only some key parameters, which can be extracted from appendix A, were contemplated when selecting random variables for mutation and creation of new individuals.

7. Results

This chapter contains all the results from before and after evolution for each of the classifiers. To obtain the values before evolution, each algorithm has been run 5 times (each time selecting a random sample of data), and the mean and standard deviation of the 5 iterations have been calculated. Furthermore, for the best iterations, the confusion matrix, the ROC curve and AUC have been calculated.

Thenceforth, the algorithm underwent *evolution*, and a evolution graph is presented for the best result after 7 generations. The evolution was measured by both the mean accuracy and the accuracy of the best individual of the generation (orange and green series, respectively). The evolution graphs present four series: the mean and best accuracy (with validation set), the value of accuracy before evolution (test set) and the accuracy of the best individual after evolution (test set).

After the processing done in chapter 4, the entire dataset was about 66000 samples long, which was too big for an average computer to execute in reasonable times. As an alternative, and for the algorithm to run smoothly, random selection of samples was performed, so only part of the total set was selected (at random) each time the programme was run. Only 0.3 of the total sample was processed, meaning the active sample set was about 20000 entries long, which was further divided into sub-samples for **cross validation**. Initially, (for "without evolution" algorithms) only two sets were needed (train and test) and for the genetic algorithm, a third set was added (validation). Table 7.1 shows the approximate sizes for the sets. Important to note that the size of the train test has changed from one algorithm to the other, due to the need of the third sub-set. Tests

	without evolution	evolution
Train	14000	8000
Test	6000	6000
Validation	-	6000
Total	20000	20000

Table 7.1.: Sub-sets sizes

were made to see which alternative returned the best outcome: increase the size of the taken sample from the original set or keep the size but alter the sub-set sizes. Finally, the results were implemented altering the training size, as it did not seem to alter the performance greatly.

7.1. Decision Tree

Table 7.2 shows the results for the 5 iterations. As it can be seen, this algorithm presented considerably stable results and very small variation.

Run	1	2	3	4	5
Accuracy	0.6319	0.6335	0.6384	0.6299	0.6283
Mean=0.6324			**Stdev=0.0039**		

Table 7.2.: DT: Results for 5 iterations

The confusion matrix in table 7.3 shows the results for iteration number 3, which presented the best accuracy.

TEST		PREDICTED	
		FALSE	TRUE
ACTUAL	FALSE	1785	991
	TRUE	1026	1776

Accuracy = 0.6384

Table 7.3.: DT: Confusion matrix for test set

7.1. Decision Tree

With only the default values, the DT classifier returned good results, as can be seen in Figure 7.1, showing an AUC of 0.62. Recalling figure 3.4, a value of AUC above 0.5 means the algorithm is capable of doing classification, and the better values are near 1.0.

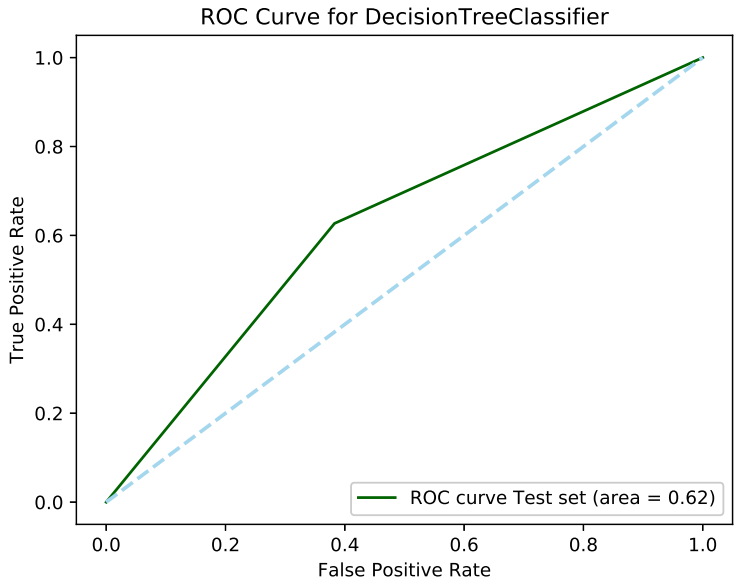


Figure 7.1.: DT: ROC curve

Aside from the ROC graph, DT provides a decision graph, presented in figure 7.2, showing the paths that could be taken when classifying an instance. As it could be predicted form initial analysis and correlation of attributes, shown in figure 4.1, the attribute *"ap_hi"*, namely *systolic pressure* is the column that provides most information about the sample due to its high correlation with the target value. Thus it comes as no surprise that this same value is the first node in the decision tree, and the most significant when making decisions, as seen on the graph.

Figure 7.3 shows the evolution undergone through 7 generations. As it can be seen, the mean and best accuracies are very similar, and the algorithm presents improvement throughout generations. Furthermore, the results are better than those obtained without evolution (from table 7.3).

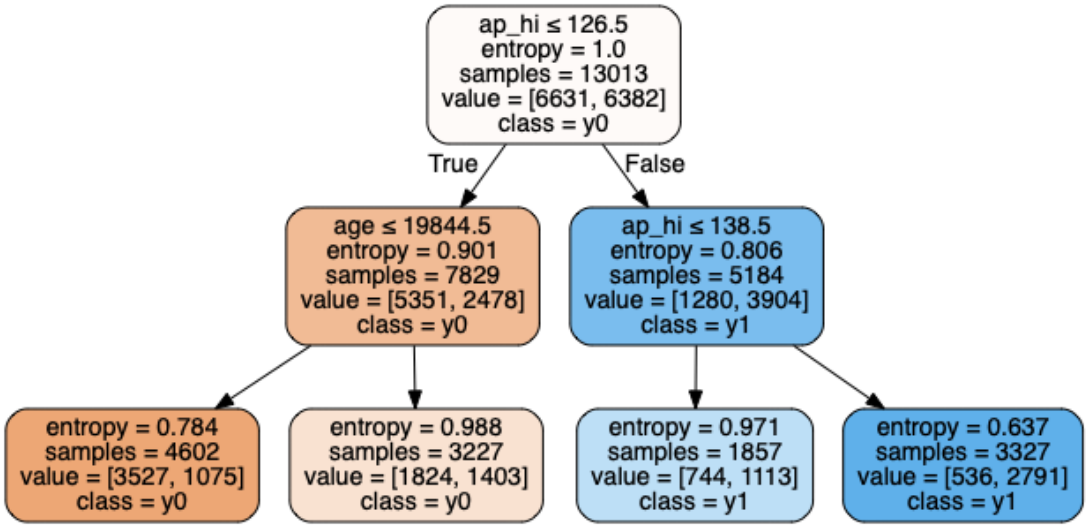


Figure 7.2.: DT: Decision graph

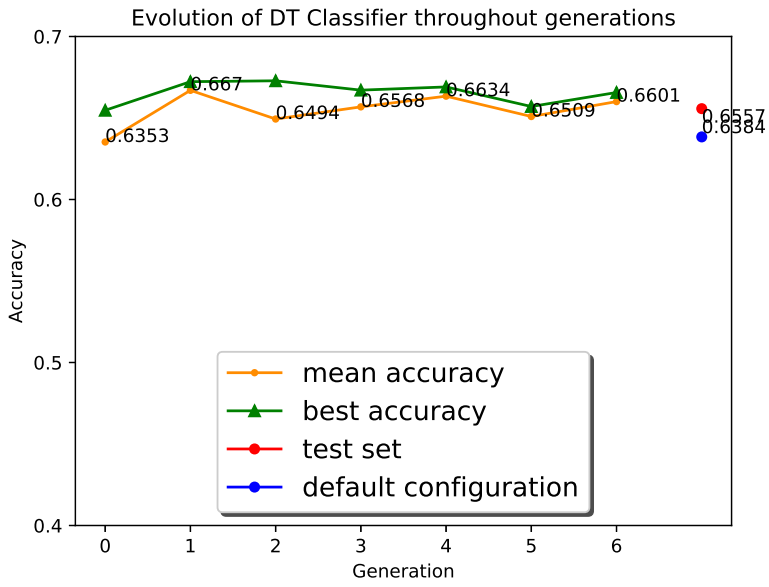


Figure 7.3.: DT: Evolution graph

7.2. Random Forest

Table 7.4 shows the results after 5 repetitions, together with the mean and stantard deviation.

Run	1	2	3	4	5
Accuracy	0.7085	0.7200	0.7094	0.7209	0.7148
Mean=0.7166			**Stdev=0.0089**		

Table 7.4.: RF: Results for 5 iterations

Table 7.5 shows the best results provided by this algorithm (from the fourth iteration), and figure 7.4 shows the results for the test set, plotted in a ROC curve. As expected, the curve shows much better results than a single Decision Tree.

TEST		PREDICTED	
		FALSE	TRUE
ACTUAL	FALSE	1234	689
	TRUE	868	1887

Accuracy = 0.7209

Table 7.5.: RF: Confusion matrix for test set

Both the shape of the curve and the value of AUC indicate good outcome from RF: the curve shows a very similar shape of that presented in figure 3.4b. It was expected that RF performed better than an individual DT, due to the own nature of the algorithm.

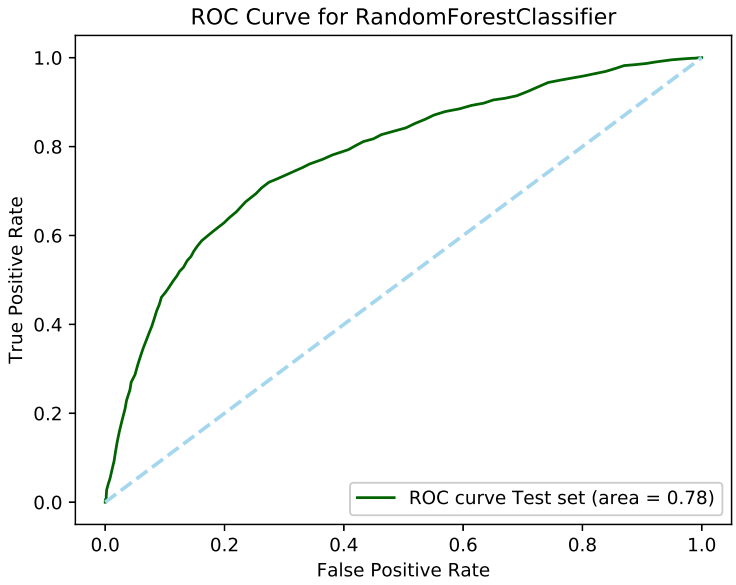


Figure 7.4.: RF: ROC curve

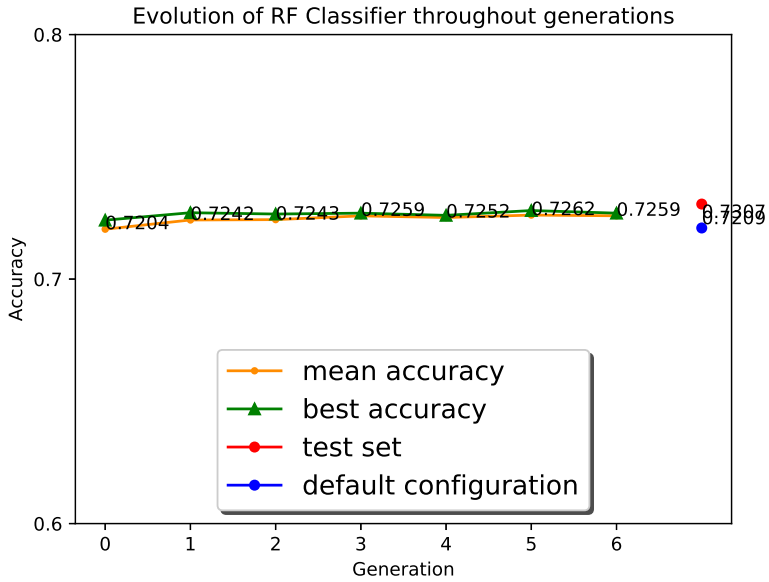


Figure 7.5.: RF: Evolution graph

The evolution graph is presented in figure 7.5. In this case, the mean and the best accuracies are identical, indicating that there was little or no variation between the indi-

viduals of the same generation. Nonetheless, there is improvement after evolution. Before evolution, the best obtained accuracy was 0.7209, and after, the best was 0.7322.

7.3. Support Vector Machine

The results obtained from executing this algorithm 5 times are presented in 7.6 and the confusion matrix for the best iteration (first) is shown in table 7.7.

Run	1	2	3	4	5
Accuracy	0.6090	0.5986	0.6034	0.5989	0.5948
Mean=0.6009			**Stdev=0.0054**		

Table 7.6.: SVM: Results for 5 iterations

TEST		PREDICTED	
		FALSE	TRUE
ACTUAL	FALSE	1510	1270
	TRUE	911	1887

Accuracy = 0.6090

Table 7.7.: SVM: Confusion Matrix for test set

Figure 7.6 shows the results for the test set, plotted in a ROC curve. As shown in the AUC value and the shape of the curve, the performance of this algorithm is good, meaning it can somewhat classify instances.

Figure 7.7 shows the little variation undergone by the SVM algorithm through the generations. This example shows in fact a decrease in the accuracy after the evolution, suggesting the need of reviewing the parameters being modified and the constants included.

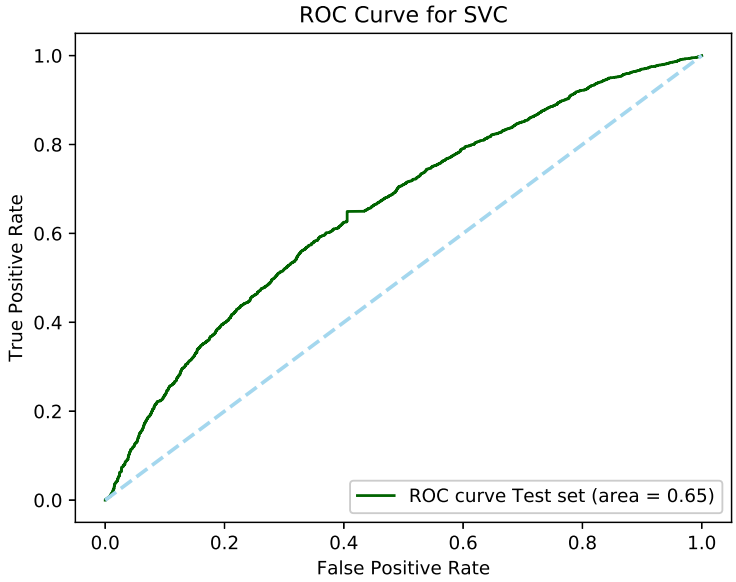


Figure 7.6.: SVM: ROC curve

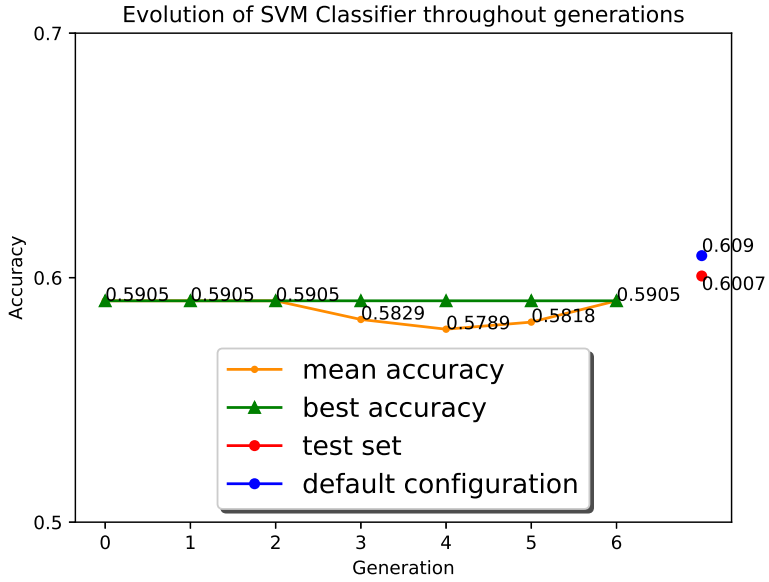


Figure 7.7.: SVM: Evolution graph

7.4. K Nearest Neigbour

Table A.4 shows all the parameters this classifier accepts, their value types and the default values. The results for the five iterations of this algorithm are shown in table 7.8 and the confusion matrix for the best iteration (second one) is shown in table 7.9

Run	1	2	3	4	5
Accuracy	0.5497	0.5611	0.5522	0.5446	0.5510
Mean=0.5517			**Stdev=0.0060**		

Table 7.8.: KNN: Results for 5 iterations

TEST		PREDICTED	
		FALSE	TRUE
ACTUAL	FALSE	1603	1220
	TRUE	1228	1527

Accuracy = 0.5611

Table 7.9.: KNN: Confusion Matrix for test set

Figure 7.8 shows the area under curve for the sklearn knn classifier. The AUC value is just over 0.5, which means that the algorithm is capable of classification, but not to a great extent.

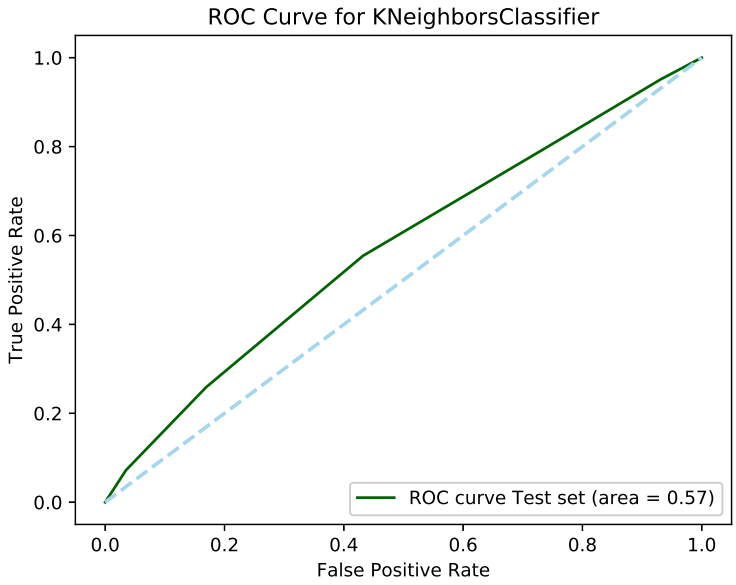


Figure 7.8.: KNN: ROC curve

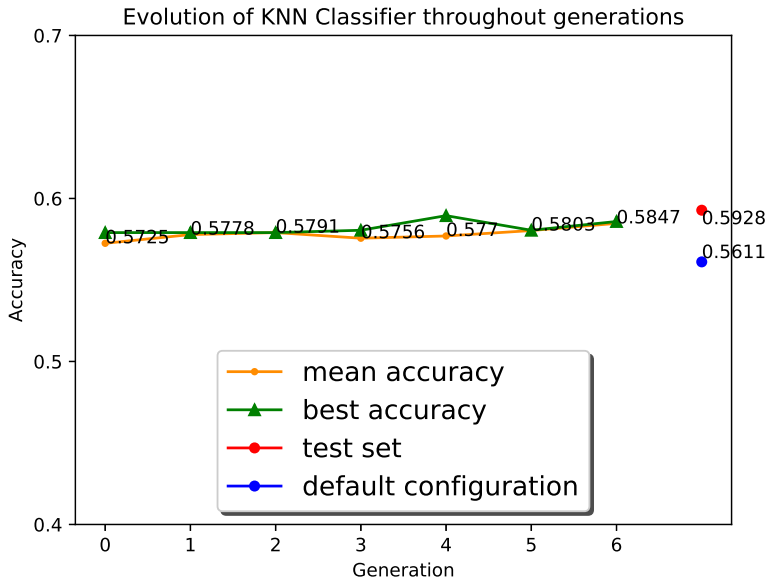


Figure 7.9.: KNN: Evolution graph

The evolution graph for the KNN algorithm is represented in figure 7.9, with the four series explained above.

7.5. Multilayer Perceptron

Run	1	2	3	4	5
Accuracy	0.5030	0.5330	0.5283	0.5816	0.5679
Mean=0.5428			**Stdev=0.0317**		

Table 7.10.: MLP: Results for 5 iterations

Table 7.10 shows the accuracy obtained in 5 iterations of the algorithm. As it can be seen, it shows much more unstable results and a larger standard deviation. Table 7.11 shows the confusion matrix for the best iteration. The ROC curve and AUC values are shown in figure 7.10.

TEST		PREDICTED	
		FALSE	TRUE
ACTUAL	FALSE	2479	315
	TRUE	2019	765

Accuracy = 0.5816

Table 7.11.: MLP: Confusion Matrix for test set

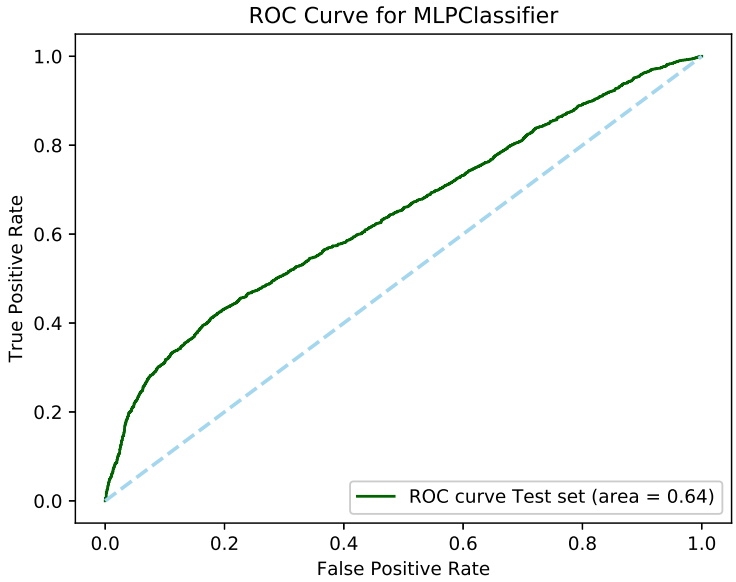


Figure 7.10.: MLP: ROC curve

The evolution of the MLP algorithm is shown in figure 7.11. As seen, it presents a significant improvement from the default configuration, although the values of both best and mean accuracies throughout generations are very unstable, meaning that the number of generations can affect greatly to the final result (if only 5 generations were taken into account, then the improvement would not have been so good, etc.).

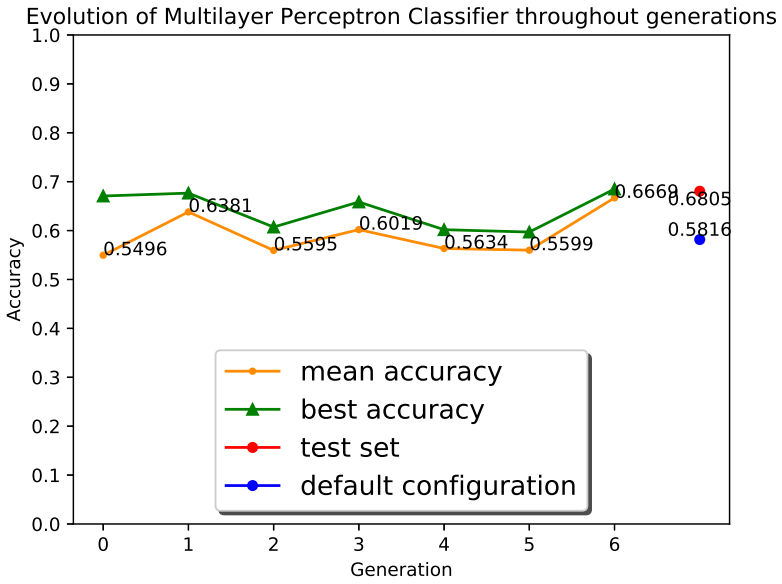


Figure 7.11.: MLP: Evolution graph

7.6. Results Overview

After the recently presented results, an overview of the programme performance was done, to have a general idea of how much the algorithms improved. Figure 7.12 shows the evolution undergone by all the algorithms throughout 7 generations. The blue series represents the best accuracy obtained from the algorithms implemented in chapter 5, while the gray series represent the accuracy of the last generation after the classifier underwent evolution, both for the test sets. As it can be seen, evolution helped the accuracy in most cases, however, some showed worse performance. In the last column of each table from

appendix A are shown the values that provided the best results.

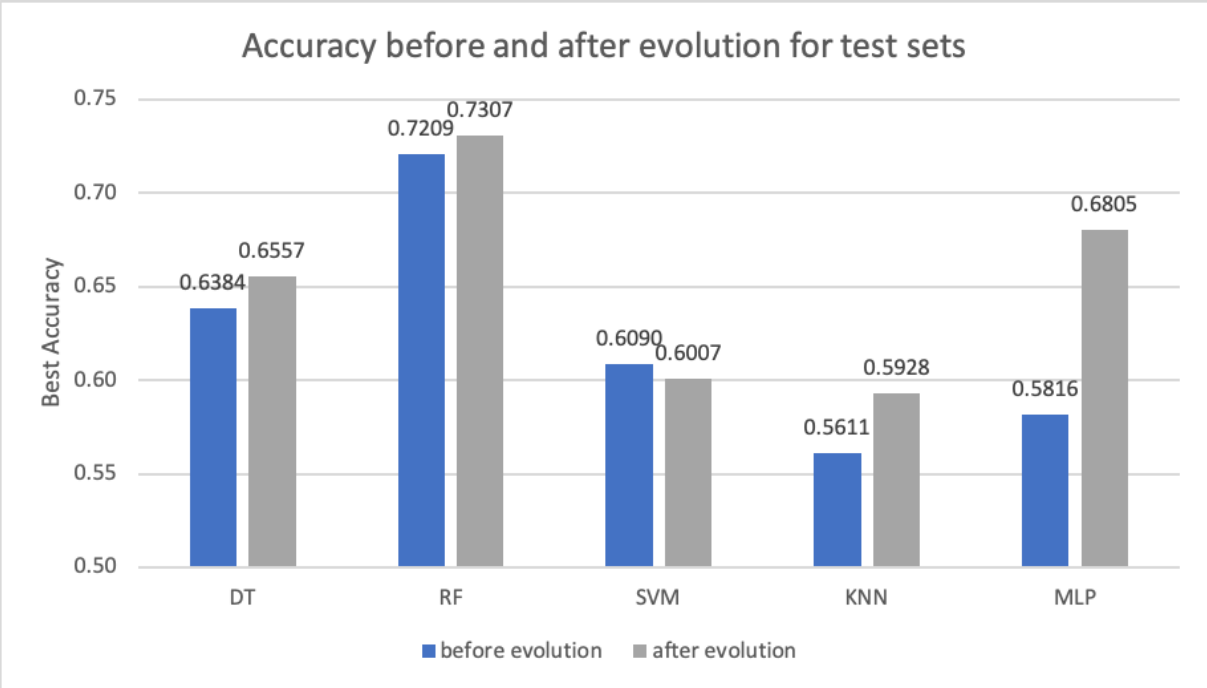


Figure 7.12.: Overview of evolution of all algorithms

The algorithm that showed the best improvement was MLP (difference of accuracies of 0.0989), although it was also the one that showed more unstable values throughout generations. Following was KNN (difference of 0.0317), DT (0.0173), RF (0.0098) and SVM (-0.0083). Despite being the fourth best improvement, RF return the overall best results, with the highest accuracy obtained of 0.7307.

8. Conclusions - Conclusiones

8.1. Conclusions (English)

This project has been a introduction to some examples of classifier algorithms and their application in the medical field. There is still a long list of improvements that could be done to this particular programme and a deeper study might help to better identify those parameters that would mean more variability in the results, without compromising the performance.

From the values obtained in chapter 7, we can see the results are heterogeneous; for the five algorithms implemented, four of them showed better accuracy after evolution (DT, RF, KNN and MLP), whilst SVM showed a decrease in its accuracy, compared to the default configuration provided by the package *sci-kit learn*. From chapter 5 we can see that implementing the classifiers is done very easily, but its understanding the parameters where it gets more complicated. Moreover, the results after evolution showed little difference compared to the default configuration. This is in fact one of the limitations encountered with genetic algorithms; very little variation throughout generations. When some of the key parameters were changed (such as probability of mutation, crossover, number of individuals, etc.), we came across much more variability, both increasing and decreasing the accuracy of individuals. Hence, it seemed better to have less variability while ensuring that the performance was always increasing rather than fluctuating to better and worse results.

The algorithm needs some improvement, not only no correct the two classifiers that

showed less accuracy, but also to increase the difference between the two series shown in figure 7.12. For this to happen, a much deeper analysis into the effect of each parameter must be done, to be able to detect which parameters should be considered for alterations and which could be altered jointly as subgroups rather than independently.

Some limitations were encountered regarding the availability of data. Although there are many different options available from the source www.kaggle.com, there is not enough information about the dataset itself, such as the meaning of the data or the values (for example, there is no information on which cardiovascular diseases where included in the study). A similar situation is presented with the attribute *cholesterol levels*: there are different types of cholesterol in the human body, and the dataset does not provide further information on the significance of the values, neither on the boundaries considered to divide the results into *normal*, *above normal* or *well above normal*. In the case of binary data, such as alcohol consumption or smoking habits, there is no further information on how this values were decided.

Another thing to consider is that regardless of the accuracy shown by the classifiers (both before and after evolution), this programme needs to be integrated with others source of information, since diagnosis needs to include much more information than these 12 parameters. Relevant data may include the evolution of such parameters through a period of time, or information from clinical analysis, medical images, electrocardiograms, etc. Together, all these systems create a great synergy, potentially capable of out-performing a human diagnostician, since the amount of information would be too much for a human to analyse. In this case, there may be necessary to implement an algorithm to detect which information is significant and thus reducing noise.

Besides the limitations the data presented, one really good aspect to consider of this set was the number of samples. As mentioned, large amounts of information was better for the algorithms, and this dataset provided about 66000 samples, which would have been ideal for our programme. However, a much more powerful computer was needed to be able to process that much data. As mentioned, to overcome this issue, random samples

of about one third of the dataset were taken for the algorithms. This means the results may vary when using a different computer capable of processing the entire set of samples; it would be of interest to analyse the impact the size of the set has on the results.

Furthermore, there is an ethical consideration that needs to be addressed. As it has been mentioned throughout the project, the aim of these programmes is not to replace doctor-patient relationship, but aid the medical staff when making decisions. Clinicians and AI can work together to greatly improve the decision making process, and thus the human aspect of a diagnosis is never lost. This is an entire ethical discussion that would escape the purposes of this project, but it is worth mentioning it as an aspect to always consider when developing new systems, specially in delicate topics such as health conditions.

8.2. Conclusiones (Español)

Este proyecto ha servido como pequeña introducción a alguno de los algoritmos de clasificación, y un ejemplo de aplicación en el sector de la medicina. Hay aún muchas mejoras que pueden implementarse a este programa en particular, para mejorar los resultados obtenidos: un estudio mucho más profundo de todos los parámetros de cada algoritmo serviría para aumentar positivamente la variabilidad en las distintas generaciones del algoritmo genético, sin comprometer la estabilidad.

De la información presentada en el capítulo 7, puede observarse que los resultados no son homogéneos, sino que en algunos casos se muestra mucha mejoría entre los algoritmos antes y después de la evolución, mientras que en otros el resultado es de hecho negativo, o sin mucho cambio. Para los 5 algoritmos implementados, cuatro de ellos mostraron un valor de exactitud mayor (DT, RF, KNN y MLP) y SVM mostró menor exactitud después del proceso de evolución. En el capítulo 5 se puede ver la facilidad con la que se implementan los algoritmos utilizando la librería de Python *sci-kit learn* y es en la comprensión de los parámetros y sus efectos donde se puede profundizar mucho el estudio. Adicionalmente, los resultados obtenidos después de la evolución presentaron poca

variación con respecto a los obtenidos previamente. Esta es una de las limitaciones que se detectaron a raíz de la implementación de los algoritmos genéticos: los valores varían poco a lo largo de las generaciones. Para intentar aumentar la variación, se alteraron algunos valores clave para el algoritmo, como son las probabilidades de mutación, cruce o reproducción. Sin embargo, estas alteraciones significaron más variación tanto positiva como negativamente, por lo que la evolución era más caótica y no seguía un patrón determinado. Se tomó entonces la decisión de mantener estos parámetros con sus valores originales, para evitar tanta fluctuación de la exactitud.

El algoritmo presentado necesita optimización para corregir los clasificadores que evolucionaron negativamente y aquellos que tenían poca estabilidad, y para aumentar la evolución positivamente de las series mostradas en la figura 7.12. Para conseguir esto, se necesita un análisis mucho más profundo de los parámetros, con el fin de detectar aquellos que tendrán más efecto (tanto positivo como negativo) y poder variarlos en grupos, en lugar de independientemente.

Se encontraron también algunas limitaciones en la obtención de datos para entrenar y evaluar el algoritmo. Como se ha comentado anteriormente, los algoritmos de AI necesitan una gran cantidad de datos para su correcto funcionamiento. Si bien hay muchos conjuntos de datos disponibles para su descarga libremente desde www.kaggle.com, no hay mucha información sobre los valores contenidos, su significancia o los baremos contemplados. Por ejemplo, no hay información sobre qué enfermedades fueron incluidas dentro del rango enfermedades cardiovasculares, o sobre los límites tenidos en cuenta al dividir la infromación entre *normal*, *por encima de lo normal* o *muy por encima de lo normal*, como es el caso de los niveles de colesterol o de glucosa en sangre. En el caso de los valores binarios, no hay información sobre cómo han sido medidos, es decir, no es posible saber el nivel de consumo de alcohol, tabaco o el nivel de actividad física que ha determinado que el valor sea 1 o 0.

Otro aspecto importante a considerar es que, independientemente de que el valor de exactitud sea mayor o menor, es vital que este programa o cualquier otro de predicción

de enfermedades, se integre con otras fuentes de información clínica del paciente, ya que el proceso de diagnóstico deberá incluir mucha más información que estos 12 parámetros. Dentro de la información clínica relevante puede estar la evolución de estos mismos parámetros en distintos momentos temporales, análisis clínicos, imágenes médicas, electrocardiogramas, etc. La integración de estos sistemas podría significar un enorme avance en el diagnóstico, posiblemente analizando mucha más información de la que un humano puede contemplar. En este caso, también sería necesario implementar algoritmos de reducción de variables, para eliminar la información poco significativa y el ruido.

Además de los aspectos limitantes de los datos usados, hay un aspecto muy relevante para los algoritmos inteligentes, y es el tamaño del conjunto de datos. Como se ha comentado anteriormente, estos algoritmos necesitan gran cantidad de datos, por lo cual el tamaño del conjunto obtenido, cerca de 66000 entradas, era ideal para el procesamiento. Sin embargo, al ejecutar el programa se encontró que el ordenador utilizado no tenía la capacidad para procesar esta cantidad de datos. Entonces, si este mismo programa se ejecutara en un computador capaz de procesar los ficheros, los resultados podrían varíar, y sería interesante analizar el impacto del tamaño del conjunto en los resultados de los algoritmos.

Por otra parte, es importante considerar los aspectos éticos que incluye este tipo de tecnología. Anteriormente se ha mencionado que el objetivo de este programa en ningún caso es el de reemplazar la figura del médico, sino de ayudar a la toma de decisiones. De esta manera, los profesionales sanitarios pueden trabajar conjuntamente con los programas para ofrecer un mejor servicio a los pacientes. Entrar en profundidad en estas discusiones éticas se escapa del propósito de este proyecto, aunque es importante tenerlas siempre en cuenta al trabajar con este tipo de tecnologías aplicados sobretodo a temas delicados, somo es la salud de los pacientes.

9. Future Research - Líneas Futuras

9.1. Future Research Lines (English)

Throughout the development of the project many points of improvement have been identified, in the software itself, the data and the integration with other systems. Initially, one very direct improvement that could trigger a much better clinical adaptation would be to combine this binary programme with a multi-class one, dividing the results not only by presence or absence of disease but go deeper into which type of disease, if any. It could be implemented as a second stage in prediction; for example, the first could be done with a binary classifier and the second stage with a multi-class neural network.

Another aspect that may improve the efficiency of the algorithm is reducing the number of variables in the dataset. In order to do this, a genetic algorithm would be needed, to detect which variables have minimum impact on the results, and if in fact, improves the efficiency.

Furthermore and maybe the biggest challenge regarding this project, is to overcome the need for integration with other systems. For more accurate results, in clinical environments, algorithms may need to include information from other sources and not only the single file of values worked with in this project. Then, the ideal scenario would be a system that integrates information from HIS, PACS, files from laboratories and any other type of clinical data, and provides a single outcome.

9.2. Líneas Futuras (Español)

Durante el desarrollo del proyecto han salido a la luz muchos aspectos mejorables, tanto del *software* mismo, como de los datos utilizados y los procesos de integración con otros sistemas. Inicialmente, un cambio que podría desencadenar una mejor adpatación clínica sería el de integrar este sistema binario con uno muti-clase, para que la predicción, en lugar de ser presencia o ausencia de enfermedad, indique también qué tipo de enfermedad se está prediciendo y con qué probabilidad. Si bien no se podría remplazar este algoritmo por uno que permita varias clases, ya que entonces reduciríamos enormemente el número de agloritmos posibles para utilizar, podría implementarse en conjunto, es decir, como primera etapa un algoritmo binario y, en una predicción posterior, uno multi-clase que podría ser, por ejemplo, una red neuronal que indique el tipo de enfermedad.

Se podría analizar la reducción de variables con las que trabaja el sistema, para aumentar la eficacia del mismo. Para esto, sería necesario un algoritmo genético que analice qué variables tienen el menor impacto en el resultado para saber si este cambio significaría una mejora en el algoritmo o la pérdida de esas variables tendría un efecto neto negativo.

Adicionalmente y posiblmente el mayor reto de este proyecto, sería integrarlo con otros sistemas de información clínica. Para mejores resultados, sería necesario incluir información de otros entornos clínicos, como podría ser un HIS o un sistema de análisis de imágenes médicas, analísis clínicos o cualquier otro tipo de dato clínico que pueda ser relevante.

Bibliography

American Heart Association. (2015). Cardiovascular disease and diabetes. Retrieved from https://www.heart.org/en/health-topics/diabetes/why-diabetes-matters/cardiovascular-disease--diabetes

American Heart Association. (2017). Understanding blood pressure readings. Retrieved from https://www.heart.org/en/health-topics/high-blood-pressure/understanding-blood-pressure-readings

Bettencourt-Silva, J., Mannu, G., & Iglesia, B. (2016). Visualisation of integrated patient-centric data as pathways: Enhancing electronic medical records in clinical practice. (Vol. 9605, pp. 99–124). doi:10.1007/978-3-319-50478-0_5

Chakure, A. (2020). Random forest classification and its implementation. Retrieved from https://towardsdatascience.com/random-forest-classification-and-its-implementation-d5d840dbead0

Costanzo, L. S. (2018). Cardiovascular physiology. In *Physiology* (pp. 117–188). Elsevier.

DeepAI. (2019). Multilayer perceptron. Retrieved from https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron

Gidding, S. S., & Allen, N. B. (2019). Cholesterol and atherosclerotic cardiovascular disease: A lifelong problem. *Journal of the American Heart Association*, *8*(11), e012924. doi:10.1161/JAHA.119.012924

Haenlein, M., & Kaplan, A. (2019). A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California Management Review*, *61*(4), 5–14. doi:10.1177/0008125619864925

Hoare, J. (2020). What is a decision tree? Retrieved from https://www.displayr.com/what-is-a-decision-tree/

Holzinger, A. (2016). *Machine learning for health informatics: State-of-the-art and future challenges*. doi:10.1007/978-3-319-50478-0

Joshi, P. (2017). *Artificial intelligence with python*. Packt Publishing. Retrieved from https://books.google.es/books?id=O1AoDwAAQBAJ

Lind, L., Risérus, U., & Ärnlöv, J. (2020). Impact of the definition of metabolically healthy obesity on the association with incident cardiovascular disease. *Metabolic Syndrome and Related Disorders*. PMID: 32397901. doi:10.1089/met.2020.0016

Manohar, S. (2017). *Mastering machine learning with python in six steps: A practical implementation guide to predictive data analytics using python*. Apress.

Martin, N. (2019). Artificial intelligence is being used to diagnose disease and design new drugs. Forbes Magazine. Retrieved from https://www.forbes.com/sites/nicolemartin1/2019/09/30/artificial-intelligence-is-being-used-to-diagnose-disease-and-design-new-drugs/

Mayo Foundation for Medical Education and Research. (2019a). Abdominal aortic aneurysm. Retrieved from https://www.mayoclinic.org/diseases-conditions/abdominal-aortic-aneurysm/symptoms-causes/syc-20350688

Mayo Foundation for Medical Education and Research. (2019b). Cardiomyopathy. Mayo Foundation for Medical Education and Research. Retrieved from https://www.mayoclinic.org/diseases-conditions/cardiomyopathy/symptoms-causes/syc-20370709

Narkhede, S. (2019). Understanding auc - roc curve. Retrieved from https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

NHLBI. (2020). Atherosclerosis. U.S. Department of Health, Human Services - National Heart Lung, and Blood Institute. Retrieved from https://www.nhlbi.nih.gov/health-topics/atherosclerosis

NHS. (2018). Cardiovascular disease. NHS. Retrieved from https://www.nhs.uk/conditions/cardiovascular-disease/

NHS. (2020). Coronary heart disease. NHS. Retrieved from https://www.nhs.uk/conditions/coronary-heart-disease/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Piano, M. R. (2017). Alcohol's effects on the cardiovascular system. *Alcohol research : current reviews*, 219–241. doi:101594475

Pollock, J. D. (2019). Physiology, cardiac cycle. U.S. National Library of Medicine. Retrieved from https://www.ncbi.nlm.nih.gov/books/NBK459327/

Than, K. (2018). What is darwin's theory of evolution? Purch. Retrieved from https://www.livescience.com/474-controversy-evolution-works.html

The Johns Hopkins University. (2020). Anatomy and function of the heart's electrical system. Retrieved from https://www.hopkinsmedicine.org/health/conditions-and-diseases/anatomy-and-function-of-the-hearts-electrical-system

U.S. Department of Health and Human Services. (2010). How tobacco smoke causes disease: The biology and behavioral basis for smoking-attributable disease: A report of the surgeon general. *PsycEXTRA Dataset*, 355–434. doi:10.1037/e590462011-001

Wang, F., & Preininger, A. (2019). Ai in health: State of the art, challenges, and future directions, 16–26. doi:10.1055/s-0039-1677908

WHO. (2017). National ehr system exists. World Health Organization. Retrieved from https://gateway.euro.who.int/en/indicators/ehealth_survey_84-has-a-national-ehr-system/

WHO. (2019). Cardiovascular diseases. Retrieved from https://www.who.int/health-topics/cardiovascular-diseases/

Yiu, T. (2019). Understanding random forest. Towards Data Science. Retrieved from https://towardsdatascience.com/understanding-random-forest-58381e0602d2

Yu, K.-H., Beam, A. L., & Kohane, I. S. (2018). Artificial intelligence in healthcare. *Nature Biomedical Engineering*, *2*(10), 719–731. doi:10.1038/s41551-018-0305-z

A. Parameters for Sklearn Classifiers

Table A.1.: Parameters for sklearn DT

parameter	type	description	default	values	best
criterion	str	Function to measure the quality of a split	“gini”	“gini”, “entropy”	“gini”
splitter	str	Function to measure the quality of a split	“best”	“random”	“best”
max_depth	int	Maximum depth of the tree	None	-	-
min_samples _split	int or foat	Minimum number of samples required to split an internal node	2	-	10
min_samples _leaf	int or foat	Minimum number of samples required to be at a leaf node	1	-	-
min_weight _fraction_leaf	float	Minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node	0.0	-	-
max_features	int, float or str	Number of features to consider when looking for the best split	None	“auto”, “sqrt”, “log2”	None
random_state	int or Random-State	Random number generator	None	-	-
max_leaf_nodes	int	Grow a tree with max_leaf_nodes best-first	None		

min_impurity _decrease	float	Min value for impurity decrease for a node to be split	0.0		
class_weight	list, dict or 'balanced'	Weights associated with classes	None	True / False	None
ccp_alpha	Non-negative float	Complexity parameter used for Minimal Cost-Complexity Pruning	0.0		

Table A.2.: Parameters for sklearn RF

parameter	type	description	default	values	best
n_estimators	int	Number of trees in the forest	100	-	182
criterion	str	Function to measure the quality of a split	“gini”	“gini” or “entropy”	“gini”
max_depth	int	Maximum depth of the tree	None	-	-
min_samples _split	int or foat	Minimum number of samples required to split an internal node	2	-	2
min_samples _leaf	int or foat	Minimum number of samples required to be at a leaf node	1	-	6
min_weight _fraction_leaf	float	Minimum weighted fraction of the sum total of weights of all the input samples) required to be at a leaf node	0.0	-	-
max_features	int, float or str	Number of features to consider when looking for the best split	“auto”	int, float, “auto”, “sqrt”,“log2”	“log2”
random_state	int or Random-State	Random number generator	None	-	-
max_leaf_nodes	int	Grow a tree with max_leaf_nodes best-first	None	-	-

min_impurity _decrease	float	Min value for impurity decrease for a node to be split	0.0	-	-
bootstrap	boolean	Whether bootstrap samples are used when building trees	True	True / False	-
obb_score	boolean	Whether to use out-of-bag samples to estimate the generalization accuracy	False	True / False	True
n_jobs	int	Number of jobs to run in parallel	None	-	-
random_state	int or Random-State	Controls the randomness of the bootstrapping of the samples used when building trees	None	-	-
verbose	int	Controls when fitting and predicting	0	-	-
warm_start	boolean	Reuse the solution of the previous call to fit and add more estimators to the ensemble		True / False	-
class_weight	dict or str	Weights associated with classes	None	“balanced”, callable	“balanced”
ccp_alpha	non-negative float	Complexity parameter used for Minimal Cost-Complexity Pruning	0.0		-
max_samples	int or float	If bootstrap is True, the number of samples to draw from X to train each base estimator	None		-

Table A.3.: Parameters for sklearn SVM

parameter	type	description	default	values	best
C	float	Number of neighbours to use	1.0	-	-
kernel	string	Kernel type to use with the algorithm. Must be one of “linear”, “poly”, “rbf”, “sigmoid”, “precomputed”	“rbf”	-	“rbf”
degree	int	Degree of the polynomial kernel function ‘poly’	3	-	-
gamma	str or foat	Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’.	“scale”	“scale”, “auto’	“scale”
coef0	float	Independent term in kernel function.	0.0	-	-
shrinking	boolean	Whether to use the shrinking heuristic.	True	True / False	-
probability	boolean	Whether to enable probability estimates	False	True / False	True
tol	float	Tolerance for stopping criterion	1e-3	-	-
cache_size	float	Specifies the cache size in MB	200MB	-	-
class_weight	dict or ‘balanced’	The ‘balanced’ mode uses the values of y to automatically adjust weights inversely proportional to class frequencies	None	-	-
verbose	boolean	Enable verbose output	False	True / False	-

max_iter	int	Hard limit on iterations within solver, or -1 for no limit.	-1	-	-
decision_ function_ shape	str	Whether to return a one-vs-rest (‘ovr’) or a one-vs-one (‘ovo’)	‘ovo’	‘ovo’, ‘ovr’	‘ovr’
break_ties	boolean	predict will break ties according to the confidence values of decision_function	False	True / False	-
random_state	int, Random- State or None	The seed of the pseudo random number generator used when shuffling the data for probability estimates.	None	-	-

Table A.4.: Parameters for sklearn KNN

parameter	type	description	default	values	best
n_neighbors	int	Number of neighbours to use	5	-	5
weights	str or callable	Weight functions for prediction	“uniform”	“uniform” “distance”	“uniform”
algorithm	str	Algorithm to compute the nearest neighbours	“auto”	“auto” “ball_tree” “kd_tree” “brute”	“auto”
leaf_size	int	Leaf size for trees. Can affect the speed of constructoin.	30	-	30
p	int	Power parameter for minkowski metric.	2	-	2
metric	str or callable	Distance metrics for the tree.	“minkowski”	“euclidean”, “manhattan”, “chebyshev”, “minkowski”, “wminkowski”	“euclidean”
metric_params	dict	Additional dict with arguments for metric function	None	-	None
n_jobs	int or None	Number of parallel jobs	None	-	10

Table A.5.: Parameters for sklearn MLP

parameter	type	description	default	values	best
hidden_layer _sizes	tuple	The ith element represents the number of neurons in the ith hidden layer.	(100,)	-	(215,)
activation	str	Activation function for the hidden layer.	“relu”	“identity”, “logistic”, “tanh”, “relu”	“relu”
solver	str	The solver for weight optimization	“adam”	“lbfgs”, “sgd”, “adam”	“adam”
alpha	float	L2 penalty (regularization term) parameter	0.0001	-	-
batch_size	int	Size of minibatches for stochastic optimizers	“auto”	-	-
learning_rate	str	Learning rate schedule for weight updates.	“constant”	“constant”, “invscaling”, “adaptive”	“constant”
learning_rate _init	double	The initial learning rate used. Only used when solver=’sgd’ or ‘adam’.	0.001	-	-
power_t	double	The exponent for inverse scaling learning rate. Only used when solver=’sgd’.	0.5	-	-
max_iter	int	Maximum number of iterations.	200	-	-
shuffle	boolean	Whether to shuffle samples in each iteration.	True	True / False	True

random_state	int	Determines random number generation for weights and bias initialization.	None	-	-
tol	float	Tolerance for the optimization.	1e-4	-	-
random_state	int or Random-State	Controls the randomness of the boot-strappingof the samples used when building trees	None	-	-
verbose	boolean	Controls the verbosity when fitting and predicting	False	-	-
warm_start	boolean	When set to True, reuse the solution of the previous call to fit as initialization	False	True / False	False
momentum	float	Momentum for gradient descent update. (only "sgd")	0.9	-	-
nesterovs _momentum	boolean	Whether to use Nesterov's momentum.	True	True / False	-
early_stopping	int or float	Whether to use early stopping to terminate training when validation score is not improving.	False	True / False	-
validation _fraction	float	The proportion of training data to set aside as validation set for early stopping	0.1	range(0,1)	-

beta_1	float	Exponential decay rate for estimates of first moment vector. Only used when solver="adam"	0.9	range(0,1)	-
beta_2	float	Exponential decay rate for estimates of second moment vector. Only used when solver="adam"	0.999	range(0,1)	-
epsilon	float	Value for numerical stability in adam.	1e-8	-	-
n_iter_no _change	int	Maximum number of epochs to not meet "tol" improvement	10	-	-
max_fun	int	Maximum number of loss function calls. Only used when solver="lbfgs".	15000	-	-

B. Python Classes

Main.py

```
import Population
import random
import numpy as np
import pandas as pd
from matplotlib import cm, pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import metrics

TARGET_ACCURACY = 1.00
N_GENERATIONS = 7
RATIO_SELECTED = 0.8
PROBAB_CROSSING, PROBAB_MUTATING = 0.4, 0.3
RANDOM_SAMPLE_FRAC = 0.3


def get_data_csv(file_name):
    data = pd.read_csv('PATH_TO_FILE' + file_name + '.csv')
    dataframe = data.sample(frac=RANDOM_SAMPLE_FRAC)
    x = np.array(dataframe.drop(['target'], 1))
    y = dataframe['target'].values
    return x, y


```

```
24 def print_confusion_matrix(i, y_true, y_pred):
25     print("\n => Individual: ", i.get_accuracy())
26     print("Confusion matrix :")
27     print(metrics.confusion_matrix(y_true, y_pred))
28 
29 
30 def print_generation_graph(mean, best_inds, clf, acc_test):
31     # acc is a list of vectors, each vector has all accuracies of its
    generation
32     # knn = 0.5527, dt= 0.7090, rf = 0.7092 , svm = 0.6013, ann=5834
33     default = dict([('DT Classifier', 0.6384), ('RF Classifier', 0.7209), (
    'SVM Classifier', 0.6090), ('KNN Classifier', 0.5611), ('Multilayer
    Perceptron Classifier', 0.5816)])
34     gen = list(range(0, len(mean)))
35     plt.figure()
36     plt.plot(gen, mean, color='darkorange', marker='.', label='mean
    accuracy')
37     plt.plot(gen, best_inds, color='green', marker='^', label='best
    accuracy')
38     plt.plot(N_GENERATIONS, acc_test, marker='o', markersize=5, color='red'
    , label='test set')
39     plt.text(N_GENERATIONS, acc_test, str(round(acc_test, 4)), ha='left',
    va='top')
40     plt.plot(N_GENERATIONS, default[clf], marker='o', markersize=5, color='
    blue', label='default configuration')
41     plt.text(N_GENERATIONS, default[clf], str(default[clf]), ha='left', va=
    'bottom')
42     for a, b in zip(gen, mean):
43         plt.text(a, b, str(round(b, 4)))
44     plt.xticks(gen)
45     plt.yticks(list([x * 0.1 for x in range(5, 8)]))
46     plt.xlabel('Generation')
47     plt.ylabel('Accuracy')
48     plt.legend(loc='lower center', shadow=True, fontsize='x-large')
```

```
49      plt.title('Evolution of %s throughout generations' % clf)
50      plt.savefig('../graphs/gen_graphs_new/gen_graph.eps', format='eps')
51      plt.show()
52
53
54  if __name__ == "__main__":
55      X, y = get_data_csv('cardio_train3')
56      X_train, X_2, y_train, y_2 = train_test_split(X, y, test_size=0.6,
        random_state=1)
57      X_test, X_val, y_test, y_val = train_test_split(X_2, y_2, test_size
        =0.5, random_state=1)
58
59      population = Population.Population()
60      mean_accuracies = []
61      best_ind_array = []
62
63      print('\nStarting the evolution process')
64      # get first generation, created 100% at random
65      # population is an array of N_INDIVIDUALS instances of the class KnnGA
66
67      for g in range(N_GENERATIONS):
68          print("\n===== Generation", g)
69          population.mate()
70          for p in population.get_individuals():
71              if random.random() < PROBAB_MUTATING:
72                  p.mutate()
73              # with PROBAB_CROSSING, cross two individuals
74              if random.random() < PROBAB_CROSSING:
75                  population.cross_over()
76              p.fit(X_train, y_train)
77              p.predict(X_val, y_val)
78              # print_confusion_matrix(p, y_val, p.get_predicted_values())
79          # selection_size = int(RATIO_SELECTED * )
```

```
80          best_inds = population.get_n_best_individuals(population.
    get_n_individuals())
81          # offpring = population.clone(best_inds)
82          population.set_individuals(best_inds)
83          best_ind = population.get_n_best_individuals(1)
84          best_ind_array.append(best_ind[0].get_accuracy())
85          mean_accuracies.append(population.get_mean_accuracy())
86
87      print("\n==== End of evolution")
88      for p in population.get_individuals():
89          p.predict(X_test, y_test)
90          y_pred = p.get_predicted_values()
91      best_ind = population.get_n_best_individuals(1)[0]
92      print(best_ind.get_parameters())
93      print_confusion_matrix(best_ind, y_test, y_pred)
94      print_generation_graph(mean_accuracies, best_ind_array, population.
    get_classifier_type(), population.get_mean_accuracy())
95      exit()
```

Listing B.1: GA Main

Population.py

```
1  # change following line to KnnGA, SvmGA, DtGA, AnnGA accordingly
2  import RfGA as clf
3  import random
4  import statistics
5
6  N_INDIVIDUALS = 5
7
8
9  class Population:
10     individuals = []
```

```
    n_individuals = N_INDIVIDUALS
    classifier_type = ''
    mean_accuracy = 0.0

    def __init__(self):
        self.individuals = self.get_random_generation()

    def get_n_individuals(self):
        return self.n_individuals

    def set_n_individuals(self, n):
        self.n_individuals = n

    def set_classifier(self, clf):
        self.classifier_type = clf

    def set_individuals(self, inds):
        self.individuals = inds

    def get_individuals(self):
        return self.individuals

    def get_random_generation(self):
        inds = []
        for i in range(self.n_individuals):
            inds.append(clf.Classifier())
        self.classifier_type = inds[0].get_name()
        return inds

    def get_classifier_type(self):
        return self.classifier_type

    def get_n_best_individuals(self, n):
        self.individuals.sort(key=lambda x: x.get_accuracy(), reverse=True)
```

```
45          inds = self.individuals[0:n]
46          return inds
47
48      def cross_over(self):
49          key = clf.get_random_key(True)
50          inds = self.get_n_random_individuals(2)
51          ind1 = inds[0]
52          ind2 = inds[1]
53          temp = ind1.get_parameter(key)
54          ind1.set_parameter(key, ind2.get_parameter(key))
55          ind2.set_parameter(key, temp)
56
57      def mate(self):
58          for i in range(0, 3):
59              inds = self.get_n_best_individuals(2)
60              ind1 = inds[0]
61              ind1_params = ind1.get_parameters()
62              ind2_params = inds[1].get_parameters()
63              child_params = dict()
64              keys = clf.get_random_key()
65              for k in keys:
66                  child_params[k] = random.choice([ind1_params[k],
    ind2_params[k]])
67              child = clf.Classifier()
68              child.set_parameters(child_params)
69              self.individuals.append(child)
70
71      def get_n_random_individuals(self, n):
72          inds = list()
73          for i in range(n):
74              inds.append(self.individuals[random.randint(0, len(self.
    individuals) - 1)])
75          return inds
76
```

```
77      def clone(self, partial_generation):
78          full_generation = []
79          i = 0
80          while len(full_generation) < self.n_individuals:
81              full_generation.append(partial_generation[i])
82              i = 0 if i == len(partial_generation) - 2 else i + 1
83          return full_generation
84
85      def get_mean_accuracy(self):
86          s = 0.0
87          for i in self.individuals:
88              s += i.get_accuracy()
89          return s / len(self.individuals)
90
91      def get_all_accuracies(self):
92          accuracies = []
93          for i in self.individuals:
94              accuracies.append(i.get_accuracy())
95          return accuracies
```

Listing B.2: GA Population

GenericClassifier.py

```
1 import random
2 from sklearn import metrics
3
4
5 def keys_to_array(params):
6     parameters = list()
7     for key in params:
8         parameters.append(key)
9     return parameters
```

```
10
11
12 class GenericClassifier:
13     name = ""
14     classifier = ""
15     parameters = dict()
16     accuracy = 0.0
17     predicted_values = []
18
19     def __init__(self):
20         self.accuracy = 0.0
21
22     # ==== GET methods ======
23     def get_name(self):
24         return self.name
25
26     def get_parameters(self):
27         return self.parameters
28
29     def get_accuracy(self):
30         return self.accuracy
31
32     def get_parameter(self, key):
33         return self.parameters.get(key)
34
35     def get_been_predicted(self):
36         return self.been_predicted
37
38     def get_predicted_values(self):
39         return self.predicted_values
40
41     # ==== SET methods ======
42
43     def set_parameters(self, params):
```

```
44          self.parameters = params
45          self.classifier.set_params(**params)
46
47      def set_parameter(self, key, val):
48          self.parameters[key] = val
49
50      def set_accuracy(self, acc):
51          self.accuracy = acc
52
53      # ====== OTHER METHODS ====
54
55      def fit(self, x_tr, y_tr):
56          self.classifier.fit(x_tr, y_tr)
57
58      def predict(self, x_test, y_test):
59          prediction = self.classifier.predict(x_test)
60          self.accuracy = metrics.accuracy_score(y_test, prediction)
61          self.predicted_values = prediction
62
63      def get_parameter_keys(self):
64          return list(self.parameters.keys())
```

Listing B.3: GA Generic Classifier

The following class is presented as an example of the 5 classes that inherit from *Generic Classifier*. The five classes follow the same structure and only the parameters will vary, all obtained from *Sci-kit learn*.

KnnGA.py

```
1 import random
2 from GenericClassifier import GenericClassifier
3 from sklearn import neighbors as knn
4 from sklearn import metrics
```

```
5
6
7  def get_random_parameters(key=False):
8      # returns a dictionary with all the parameters for the knn classifier,
       generated at random
9      param_dict = dict([
10         ('n_neighbors', random.randint(5, 100)),
11         ('weights', random.choice(["uniform", "distance"])),
12         ('algorithm', random.choice(["auto", "ball_tree", "kd_tree", "brute
       "])),
13         ('leaf_size', random.randint(1, 28)),
14         ('metric', random.choice(["euclidean", "manhattan", "chebyshev", "
       minkowski"])),
15         ('metric_params', None),
16         ('n_jobs', None),
17         ('p', 2)
18     ])
19     if not key:
20         return param_dict
21     else:
22         return param_dict[key]
23
24
25 def get_random_key(get_key=False):
26     values = ['n_neighbors', 'weights', 'algorithm', 'leaf_size', 'metric']
27     return random.choice(values) if get_key else values
28
29
30 def get_random_generation(n_individuals):
31     # create the first random generation. returns an array of random KnnGA
       objects
32     generation = []
33     for i in range(n_individuals):
34         generation.append(Classifier())
```

```
35      return generation
36
37
38  class Classifier(GenericClassifier):
39      name = "KNN Classifier"
40      classifier = knn.KNeighborsClassifier()
41
42      def __init__(self):
43          self.parameters = get_random_parameters()
44          self.classifier.set_params(**self.parameters)
45          self.been_predicted = False
46          self.predicted_values = []
47          self.accuracy = 0.0
48          GenericClassifier.__init__(self)
49
50      def mutate(self):
51          params = self.get_parameters()
52          key = get_random_key(True)
53          params[key] = get_random_parameters(key)
54          self.set_parameters(params)
55          self.been_predicted = False
```

Listing B.4: GA KNN Classifier

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA